BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2020 Bc. Andrej Dobrík

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS
ÚSTAV TELEKOMUNIKACÍ

CAPTURING CYBER-THREATS OF INDUSTRIAL SYSTEMS
ZACHYTÁVÁNÍ KYBERNETICKÝCH HROZEB INDUSTRIÁLNÍCH SYSTÉMŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Andrej Dobrík**
AUTOR PRÁCE

SUPERVISOR — **Ing. Radek Fujdiak, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Communications and Informatics**

Department of Telecommunications

Student: Bc. Andrej Dobrík ***ID:*** 183628

Year of study: 2 ***Academic year:*** 2019/20

TITLE OF THESIS:

Capturing cyber-threats of industrial systems

INSTRUCTION:

The student will have to describe the issues of industrial networks, used communication protocols, vulnerabilities, currently available tools such as honeypots for industrial systems, along with tools for creating honeypots with high interaction, including simulation/emulation tools for industrial communication protocols. The analytical part will result in a selection of components for creating honeypot with high interaction and the possibility of simulation (emulation) of at least two communication industry protocols. Within the honeypot itself, the possibilities of honeypot detection, hiding techniques, but also sandboxing, logging methods, and other techniques increasing the security, will also be explored. As part of the detection, the created honeypot will be firstly tested in a private network and then deployed in the public network. After evaluating the data, the honeypot will be optimized and finalized into a form for easy deployment with the possibility of simple configuration (network settings, services, protocols, etc.).

RECOMMENDED LITERATURE:

[1] SPITZNER, Lance. Honeypots: tracking hackers. Reading: Addison-Wesley, 2003.

[2] MACAULAY, Tyson; SINGER, Bryan L. Cybersecurity for industrial control systems: SCADA, DCS, PLC, HMI, and SIS. Auerbach Publications, 2016.

Date of project specification: 3.2.2020 ***Deadline for submission:*** 1.6.2020

Supervisor: Ing. Radek Fujdiak, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
Subject Council chairman

ABSTRACT

With knowing that cyber attacks cost corporations billions each year ranging from attempted intrusions, distributed denial of service attacks (DDOS) to viruses and computer worms etc., comes a problem with the tools available to the system administrators have at their disposal. This master´s thesis dives into exploring one of such tools, a Honeypot. More specifically, a Honeypot solution for Industrial Control Systems. From the early historical implementations of such systems, through analysis of current solutions to a creation of a new Honeypot solution, with High-interaction nature and it subsequent deployment onto a clean Virtual Private Server, followed by analysis of the intrusions that occur during the deployment period.

KEYWORDS

Honeypot, Industrial Control Systems, Operation Technology, High-interaction, Cyber security

ABSTRACT

S vedomím že kybernetické útoky stoja korporácie každoročne miliardy, počínajúc neoprávnenými útokmi, distribuovanými útokmi odmietnutia služieb (DDOS) až po vírusy a počítačové červy atď., prichádza problém s nástrojmi, ktoré majú k dispozícii správcovia systému. Táto diplomová práca sa venuje skúmaniu jedného z takýchto nástrojov, Honeypot. Presnejšie, Honeypot zariadeniam pre priemyselné riadiace systémy. Od historicky počiatočných implementácií takýchto systémov, cez analýzu súčasných riešení až po vytvorenie nového riešenia Honeypot, s vysokou mierou interakcie a následným nasadením na nový virtuálny súkromný server, po ktorom nasleduje analýza narušení, ktoré sa vyskytnú počas obdobia nasadenia.

KEYWORDS

Honeypot, Priemyselné Riadiace Systémy, Prevádzkové Technológie, Vysoká Interakcia, Počítačová Bezpečnosť

DOBRÍK, Andrej. *Zachytávání kybernetických hrozeb industriálních systémů.*. Brno, Rok, 90 p. Master's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Advised by Ing. Radek Fujdiak, Ph.D.

ROZŠÍRENÝ ABSTRAKT

S vedomím že kybernetické útoky stoja korporácie každoročne miliardy, počínajúc neo-právnenými útokmi, distribuovanými útokmi odmietnutia služieb (DDOS) až po vírusya počítačové červy atď., prichádza problém s nástrojmi, ktoré majú k dispozícii správ-covia systému. Každý z nás je potenciálnym cieľom, dokonca aj počítače alebo iné zariadenia, ktoré obsahujú užitočné alebo cenné informácie, sa môžu použiť ako zombie zariadenia, ktoré umožňujú narušiteľovi sťahovať škodlivý softvér, napríklad trójske kone alebo počítačových červov, toto zariadenie sa využíva za účelom realizácie útoku na iné zariadenia.

Ďalším problémom, ktorému musíme každý čeliť, je to, že prostredie je v neustálom stave zmien a je stále ťažšie udržať krok s najnovšími bezpečnostnými trendmi. Táto diplomová práca sa venuje skúmaniu jedného z takýchto nástrojov,Honeypot. Presnejšie, Honeypot zariadeniam pre priemyselné riadiace systémy. Od his-toricky počiatočných implementácií takýchto systémov, cez analýzu súčasných riešeníaž po vytvorenie nového riešenia Honeypot, s vysokou mierou interakcie a následnýmnasadením na nový virtuálny súkromný server, po ktorom nasleduje analýza narušení,ktoré sa vyskytnú počas obdobia nasadenia. Táto práca bude rozdelená do troch častí, teoretickej analýzy, praktického nasadenia Honeypot systémov, ich evaluácie a následné nasadenie vlastného Honeypot systému na virtuálny privátny server.

Prvá časť je zameraná na teoretický popis a analýzu priemyselných riadiacich systémov, ich účel a prípady použitia so základnou terminológiou priemyselných riadiacich systémov. Rozdiely medzi informačnými technológiami (IT) a systémami operačných techológií (OT), po ktorých nasleduje opis komunikačných protokolov, ktoré budú implementované v tejto práci. Ďalej bude nasledovať popis možných zraniteľností a typov škodlivých útokov vykonávaných útočníkmi na priemyselné riadiacie systémy. Ďalšia časť teoretickej analýzy sa zameria na systémy Honeypot. Ich opis, krátku históriu, prípady použitia a ich korelácia s priemyselnými kontrolnými systémami. Nasledná analýza jednotlivých typov Honeypot systémov, ich kategorizácia na základe behaviorálnych čŕt a nakoniec krátky teoretický pohľad na vybrané systémy Honeypot a technológie ktoré je možné využiť pri štrukturalizácií nového Honeypot riešenie v prípade vysokej miery interakcie.

Teoretická časť práce sa v konečnom dôsledku bude zaoberať aj virtuálnym prostredím, na ktorom bude každý z vybraných Honeypot systémov emulovaný. Nasleduje proces inštalácie systémov Honeypot a ich spustenie. Ďalej budú testované scenáre, ktoré budú založené na službách, ktoré poskytuje každý z Honeypotových systémov. Prvotným cieľom praktickej časti tejto práce je praktická analýza Hypypotových systémov pre priemyselné kontrolné systémy (ICS) a praktický úvod do vybraných systémov (počiatočná inštalácia, konfigurácia prostredia a skúšobné

nasadenie), za ktorými nasledujú údaje, ktoré každý z týchto systémov poskytne a nepovinné referenčné testy zdrojov, ktoré tieto systémy využívajú. Hlavné zameranie sa však bude týkať údajov, ktoré budú poskytnuté vybranými systémami Honeypot.

Analytická časť tejto práce obsahovala popis dostupných industriálnych Honeypot riešení: PentBox, Cowrie, Conpot. Tieto riešenia boli popísané od ich využitia, inštalácie, schopností záznamov a miery interakcie s týmito zariadeniami. Následne boli vykonané základné testy jednoduchých útokov na tieto zariadenia. Počas týchto tetov boli realizované benchmark testy záťaže Honeypot riešení Cowrie a Conpot na hosťovský systém. Prvotná myšlienka bola pokračovať v rozšírení Conpot Honeypotu o ďalšie protokoly, nebolo ale jasné či by bolo možné rozšíriť toto riešenie na mieru vysokej interakcie. Preto sa od tohoto riešenia upustilo a skúmali sa technológie ktoré by mohli byť využité a následne rozšírené pre toto splnenie zadania. Taktiež bola vytvorená tabuľka, ktorá sumarizuje tieto riešenia, ích využitie s výhodami a nevýhodami.

Po zmenách vykonaných v zadaní diplomovej práce sa malo nasadenie riešenia Honeypot priemyselného riadiaceho systému s vysokou interakciou vykonať navonok, rozhodnutie bolo nasadiť ho do hostiteľskej služby VPS `Leaseweb.com`, vzhľadom na pozitívne recenzie a kvalitnú zákaznícku podporu. Nasadené riešenie ICS Honeypotu poskytuje dva implementované protokoly, Modbus TCP a EtherNet/IP beziace na platforme MiniCPS. Celý systém využíva viac častí, MiniCPS, ktorý simuluje fyzické procesy, MiniNet na simulácii topológie siete, HonSSH ako riešenie na zaznamenávanie bash aktivity a nakoniec kontajnerizáciu pomocou Docker. Ktorý izoluje útočníka pred súčasťami hostitelského stroja. Z kontajnera Docker sú útočníkom prezentované procesy bežiace z MiniCPS a jadro Ubuntu. Ak je kontajner narušený, je ľahko obnoviteľný z predtým vytvoreného obrazu kontajnera. Kontajner Docker vystavuje port SSH 2222 hlavnému hostiteľskému stroju, ku ktorému je pripojený HonSSH, ten vystavuje port 22 internetu a všetky spojenia sa ním tlačené do kontajnera.

Riešenie tohoto Honeypotu bolo testované v troch expozičných obdobiach. Hlavným cieľom prvého nasadenia bolo zvýšiť počet úspešných pripojení k Honeypotu, a to vďaka schopnosti HonSSH spoofingu pomocou hesla založeného buď na zozname hesiel v konfiguračnom súbore alebo pravdepodobnosti úspešného pripojenia. Počas tohto obdobia expozície bolo zopár zaujímavých relácií. Tieto relácie zahŕňali sťahovanie externých súborov a násilné odstránenie niekoľkých adresárov OS. Druhé obdobie zahŕňalo útoky, ktoré sa snažili tlačiť údaje o ťažbe dát z externých webových stránok, sťahovanie skriptov a zmeny povolení. Tretia expozičná doba zahaŕňala útoky z dvoch predchádzajúcich období. Nevyskytli sa žiadne útoky, ktoré by zahŕňali zámer zneužitia procesov ICS. Praktická časť taktiež ob-

sahuje výpisy jednotlicých útokov, ktoré boli realizované počas expozičných periód. Vizualizácia výsledkov tejto práce mala prebehnúť pomocou vynesenia záznamov z IDS riešenia Suricata do webového riešenia Grafana. Bolo zaznamenané okolo 50 GB záznamových dát. Nastal ale problém, ktorý pravdepodobne spôsobila súčasť GeoIP. Taktiež nastala skutočnosť že lokalizačné dáta ktoré boli zaznamenané pomocou HonSSH, niekedy neobsahovali lokáciu iba IP adresu. Preto sa tieto IP adresy pretlačili cez `IPstack.com`, ktorý v niektorých prípadoch vypísal rozdielne lokácie v porovnaní s HonSSH. Na dosiahnutie vizualizácie útokov, bol vytvorený `.xlsx` súbor obsahujúci lokácie útokov, tento súbor bol následne uploadnutý do web riešenia `Google My Maps`.

Dospelo sa k záveru, po rozšírení niektorých aspektov rozpoznania že sa jedná o cielene zranitelné zariadenie, že toto Honeypot riešenie slúži svojmu účelu aj pre nezaznamenané relácie útočiace na ICS procesy, toto tvrdenie sa ale mohlo zmeniť ak by boli expozičné doby dlhšie. Napriek tomu sa Honeypot opísaný v tejto práci javí ako spoľahlivé riešenie a užitočný doplnok k sieti, najmä vďaka schopnostiam záznamu aktivity.

DECLARATION

I declare that I have written the Master's Thesis titled "Zachytávání kybernetických hrozeb industriálních systémů." independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Radekovi Fujdiakovi Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Contents

List of Figures

List of Tables

Listings

Introduction

With knowing that cyber attacks cost corporations billions each year ranging from attempted intrusions, distributed denial of service attacks (DDOS) to viruses and computer worms etc., comes a problem with the tools available to the system administrators have at their disposal [1]. Moreover, these tools are primarily on the defensive end, such as additional firewalls or intrusion detection systems. Furthermore, everyone is a potential target, even computers or other devices that bear zero-to-none useful or valuable information on them, could be used as a zombie devices that allow the intruder to download malicious software, such as worms or trojan horse, through this device, in order to implement the attack on other desired devices [2]. Another problem everyone has to face is that the environment is in a constant state of change and it is getting significantly difficult to keep up with the newest security trends. The main focus of this thesis will be on such security systems resource used in cyber attack prevention, a Honeypot. This thesis will be divided into two parts, theoretical analysis and practical deployment of Honeypot systems.

The first part will focus on the theoretical description and analysis of Industrial Control Systems, their purpose and use cases with basic terminology of the Industrial Control Systems. Differences between Information Technology (IT) and the Industrial Control Systems followed by the description of communication protocols and possible vulnerabilities and types of malicious attacks carried out by the intruders.

Next section of the theoretical analysis will focus on the Honeypot systems. Their description, brief history, use case and their correlation with the Industrial Control Systems. Followed by analysis of the individual types of Honeypot systems, their categorization based on behavioral traits and finally a brief theoretical over view of selected Honeypot systems.

Lastly, the theoretical part of this thesis will comment on the virtual environment on which will each of the selected Honeypot systems be emulated on and the resources dedicated to each environment. Following with the installation process of each Honeypot system and it's triggering. Furthermore, there will be test scenarios conducted, based on the services each of the Honeypot systems provides.

The initial aim of the practical part of this thesis is the practical analysis of Honeypot systems for Industrial Control Systems (ICS) and a practical introduction to the selected systems (initial installation, configuration of the environment and test deployment), followed by the data that each of these systems will provide and optional benchmark tests of the resources that these systems take up. The main focus however, will be on the data provided by the selected Honeypot systems.

1 Industrial systems

Supervisory control and data acquisition systems (SCADA), Process control Systems (PCS), Distributed control systems (DCS), Operation system (OT) a rather mouthful of applied terms connected with the the Industrial Control Systems (ICS) which is essentially structured by these and other various components. These control systems can be found in industries with demanding and critical environment such as electrical power-plants, pipelines, dams, nuclear power-plants, etc. All of these environments are incredibly sensitive and demanding in connection with ensuring smooth functionality of everyday networked society.

The main function of systems such as Process control Systems (PCS) and Distributed control systems (DCS) is to react accordingly to the changes in the production environment based on evaluated parameters. Industrial Control System (ICS) harvest information from various end-devices that measure the production process status. Supervisory control and data acquisition systems (SCADA), Process control Systems (PCS), Distributed control systems (DCS), etc. evaluate the gathered data and react accordingly, either by informing the system operators about the change or by automated response and action, to the changes in the system based on logic rules that trigger such events. Considering the importance of security, Operating Systems (OT) do not differ much from the Information Systems, despite that being said, there are few differences centered around these systems.

When it comes to security failures however, Industrial Control Systems in contrast to Information Systems, often bear immediate physical consequences. ICS security failures are generally initiated by anomaly behaviors or failures in maintenance, translating into maintenance trips and stoppages of production processes, complicating the diagnosis procedure.

Some of the older ICS systems are significantly more vulnerable to security breaches, for their inability to be upgraded or patched due to the outdated technology implemented, with some system assets being stretched over a significant portion of landscape with required remote access or the inability to implement basic protection measures such as firewall. ICS is also prone to multitude of security threats, for it operates with atypical network protocols, hard-programmed safety measure commands such as alarms or event traffic [3]. Following list describes some of the fundamental terms in Industrial Control System security [4].

- ***IT*** - Information Technology. Software and hardware with main functions of gathering, storing and transmitting the information data.
- ***OT*** - Operational Technology. Software and hardware with main functions of monitoring, evaluating and adjusting changes in physical devices directly.
- ***CNI*** - Critical National Infrastructure. Systems (including both IT and OT) and assets that are essential for the function of society.
- ***ICS*** - Industrial Control Systems. Computer systems that are monitoring and operating complex physical and manual processes. Which we can divide into two categories: Supervisory Control and Data Acquisition systems (SCADA) and Distributed Control Systems (DCS).
- ***SCADA*** - Supervisory Control and Data Acquisition systems. Systems used for processing the communication flow and management of system operations and processes. Spanning over WAN (Wide Area Network).
- ***DCS*** - Distributed Control Systems. Autonomous controllers spanning over the whole system with no central supervisory unit.
- ***HMI*** - Human Machine Interface. Interaction interface enabling the users to control the systems.
- ***PLC*** - Programmable Logic Controller. Small real-time operating evaluation and control units.
- ***IIoT*** - Industrial Internet of Things. Term used in progressively increasing demand of connecting the operational systems to the internet.

1.1 Information Technology versus Operational Technology

Essentially IT systems focus on the development, interpretation, maintenance and the actual use of the computer systems, with the help of software and networking systems to transmit, process and share data [5]. In contrast to IT, Operational Technology (OT) primarily deal with the detection, control and the monitoring of the physical devices and processes implemented in the industrial systems by using controllers, sensors, actuators, Programmable Logic Controllers (PLC), Discrete Process Control systems (DPC), Building Automation Systems (BAS), Process Control Dynamics (PCD), etc.

Even though, these systems have different usage and requirements as far as security, operations and maintenance are concerned, by linking these complementary system we increase the automation capabilities of the system as a whole [6]. The following Fig. 1.1 describes the partitions forming the Operational Technology systems.

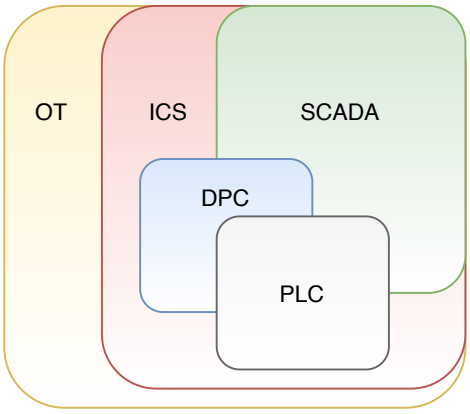

Fig. 1.1: Operational Technology partitions.

1.2 Communication protocols in OT

As mentioned earlier, due to the differences in Information Technology (IT) and Operational Technology (OT), common IT communication protocols can not be used in OT for data transfer and communication. Unlike common IT systems using standardized TCP/IP protocols, Operational Technology (OT) are distinguishable by significant variation in communication protocols that they use. In traditional OT systems, there is a range of widely used protocols such as ***Modbus***, ***HART***, ***PROFIBUS***, ***PROFINET***, ***RS-232*** and ***RS-458***, ***DNP3***, ***CIP***, ***ICCP***, ***TASE 2.0***, ***BACnet***, ***FOUNDATION Fieldbus***, etc. Following text will describe these protocols in more detail [7]. This diploma thesis will be implementing communication of two Industrial Control System protocols, Modbus protocol and EtherNet/IP (IP as industrial protocol) protocol, for the development platform MiniCPS which will be described in further section of this thesis, works with the libraries supporting these protocols. The following text will describe these protocols further. This thesis will not go in deep description of how the following protocols work, for this thesis will not deal with implementation of these protocols but rather the utilization of the respected protocols libraries, to implement processes simulating devices which communication is supported by these protocols.

Modbus is widely deployed Industrial Control Systems (ICS) serial communication protocol, also one of the oldest. It falls under the open-source category. It's communication is done with unauthenticated raw messages without any overhead, based on the request/response principle operating on the application layer. It´s origin dates to the 1979, when it was published by the `Modicon` eventually renamed to the `Schneider Electric` [8]. The Modbus protocol usage spans over multiple applications in the OT, from collecting data from sensors and actuators to SCADA

systems connecting supervisory devices with the RTU (Remote Terminal Unit).

Modbus protocol is implemented in number of variants ***Modbus RTU***, ***Modbus TCP***, ***Modbus over UDP***, ***Modbus Plus***, ***Pemex Modbus***, etc. This thesis will be using Modbus TCP by utilizing the `PyModbus` python library. Put simply, Modbus TCP utilizes TCP/IP and Ethernet to transport the structure of a Modbus message from one Modbus compatible device to another Modbus compatible device. Modbus as a data representing protocol with the combination of the networking standart TCP/IP, manifests into the Modbus TCP/IP. Moreover, the Modbus TCP/IP content can be described as an encapsulated Modbus data wrapped in the Ethernet TCP/IP. As illustrated in the Fig. 1.2, the system behind construction of how a Modbus TCP packet is structured together [9].

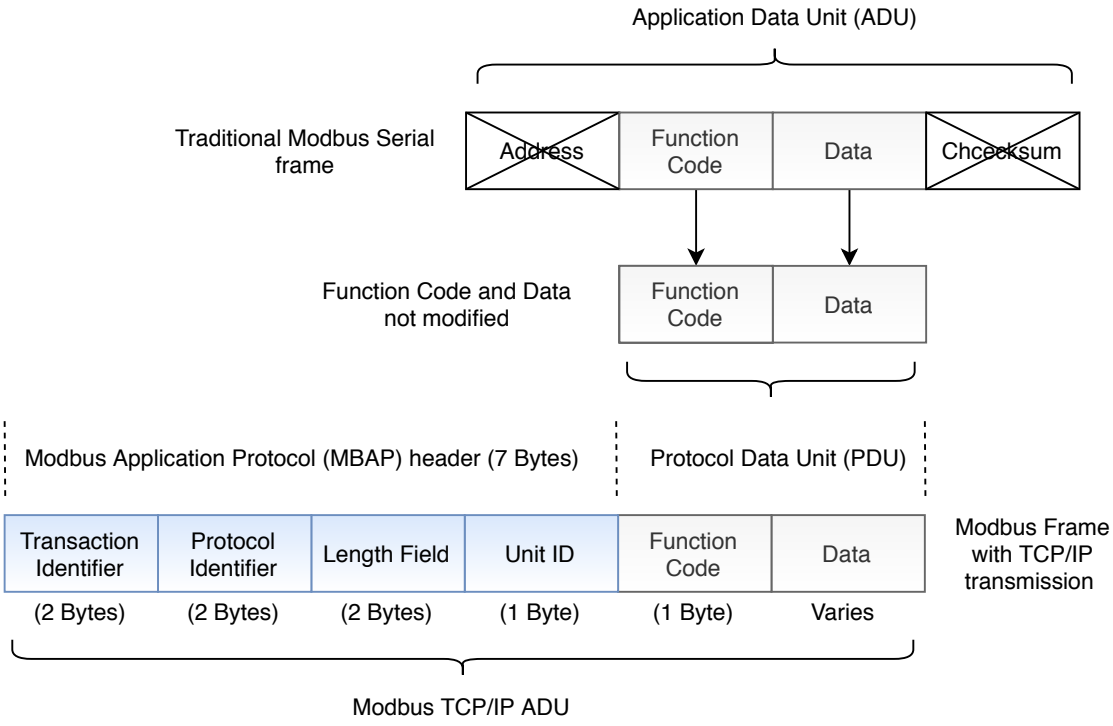

Fig. 1.2: Modbus TCP data packet construction.

From the diagram it is clear that the Modbus data, while being wrapped into the TCP/IP Application Data Unit, is not in any way modified. Moreover, the TCP/IP link layer utilizes it´s own `checksum`, so the Modbus `checksum` field is not being used to ensure the data integrity on it´s own. Another field that is substituted is the `address`, this is done in by the Modbus Application Protocol (MBAP) header, more specifically the Unit Identifier, of which the `address` becomes a part of. This is also illustrated in the Fig. 1.2. Thus the whole Application Data Unit of the Modbus TCP/IP is wrapped into the TCP frame and subsequently transferred via

a dedicated well known TCP port 502. Both servers and clients utilizing the Modbus protocol listen on this port.

The data model of the Modbus protocol operates with four data types: ***Discrete Inputs***, ***Coils*** **(Outputs)**, ***Input Registers*** **(Input Data)** and ***Holding Registers*** **(Output Data)** that are furter described in the Tab. 1.1.

Object Type	Function	Size	Address Space
Coil	read-write	1bit	00001 - 09999
Discrete Input	read-only	1 bit	10001 - 19999
Input register	read-only	16 bits	30001 - 39999
Holding register	read-write	16 bits	40001 - 49999

Tab. 1.1: Modbus protocol object types

EtherNet/IP , is a protocol implementation of the `CIP` protocol over standart Ethernet [10]. Common Industrial Protocol (CIP) was designed to automate industrial applications. CIP incorporates a set of security, synchronization and control services and messages. CIP's ease of use and integration into networks translates into it's wide use in industry. CIP was designed for integration and intercommunication with different networks. Remote attack vulnerabilities of CIP range from Denial of Service (DoS), Man in the Middle (MitM) to controller faults.

`IP` in the `EtherNet/IP` refers to `Industrial Protocol`. The Fig. 1.3 illustrates, how the EtherNet/IP protocol utilizes the TCP/IP protocol suite throught the CIP protocol over standart `IEEE 802.3`. EtherNet/IP protocol provides multiple services through CIP.

The application layer of CIP provides set of application objects along with device profiles, defining common behaviors and interfaces. Furthermore, end to end communication is provided by the CIP services. The EtherNet/IP protocol maps this communication through CIP protocol to Ethernet and TCP/IP suite, which subsequently enables communication and interoperability across multiple Ethernet devices, along with different CIP networks.

To include this protocol in this thesis Honeypot solution, the `CPPPO` (Comm. Protocol Python Parser and Originator) python library will be utilized. This library provides a subset of implemented EtherNet/IP client and server protocol. Along with this an Allen Bradley Control Logix 5561 controller subset of Tag communication is provided, supporting the read/write Tag services.

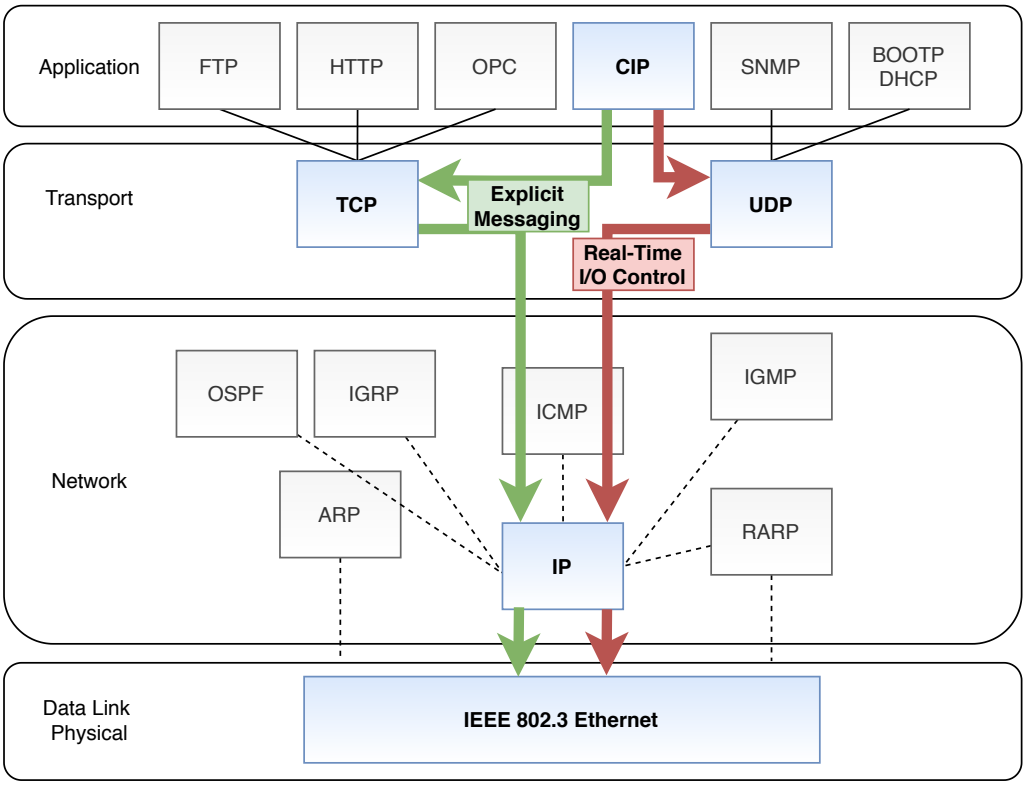

Fig. 1.3: EtherNet/IP CIP utilization through the TCP/IP suite.

1.3 Attacks in Industrial Control Systems

Cyber events concerning the Industrial Control Systems (ICS) rose significantly in 2010 with the Stuxnet malware discovery. Since this malware emersion, ICS asset operators and owners experienced several dedicated software intrusions, in the following years to come, some of which will be listed below:

- ***HAVEX*** malware spreading to several ICS-related organizations between the years 2011 and 2015
- 2014 Germany steel mill incident, attributed to a cyber activity by the German government
- 2015 and 2016 power events in the Ukraine
- ***TRISIS***, a 2017 oil and gas facility attack in Saudi Arabia
- 2017 and 2018 series of electric utilities and grid operators intrusions targeted on the UK, US and Germany, ***Shamoon3*** targeting oil, gas organizations in Middle East and Europe
- 2019 ***LockerGoga*** caused more than 52 million dollars in damages

Studies show that known ICS attacks have not only increased in more frequent occasions, but have furthermore increased in the severity, progressing from data gathering and enumerations (HAVEX) to disruptions in operation (events in Ukraine) to a potential destruction of physical assets (TRISIS, LockerGoga) [11].

Considering that prevention and counter measurement defenses are reported from the outcome of the ICS intrusions, these attacks require and rely on number of steps prior to the successful attack outcome. Moreover, the overall view of the network defence, attack steps and response actions can condense the ICS events into a sequence of steps necessary to carry out the objective of the attacker.

The stages of positive initiation, subsequent continuation and desired outcome in a form of successful network attack on the ICS focused systems are illustrated in the Fig. 1.4, the ICS Cyber Kill Chain. After realizing this structure, ICS attacks are carried out in a step by step fashion, rather than one isolated event.

Furthermore, the ultimate goal is achieved by completion of several linked objectives of the process. Based on this, the ICS network defenders must adjust to this fact as well, for different tactics can be used in various stages to penetrate the system. For instance one set of tactics can be used in the initial access part, while an entirely different tactic can be used during the ICS disruption execution. By identifying the similarities in the various stages of the ICS Cyber Kill Chain, the defenders can adjust and deploy countermeasures according to the previously established sets of attacks [12].

The ICS Cyber Kill Chain could be divided into two major stages, the first of the two stages begins with a Planning Phase where reconnaissance is the main objective, furthermore by gathering available data about the target, the attacker can exploit the weaknesses of not only the target but also the ICS technical vulnerabilities by gaining information about the operating model. Some of passive reconnaissance techniques is called footprinting, which takes advantage of the tremendous increase in information availability. Other methods include active mapping of the target's publicly available resources, activity patterning, operation system categorization or hiding in the noise of assumed internet traffic.

The second phase of the first stage is the Preparation Phase which includes Weaponization and Targeting. Weaponization is based on the modification of harmless files, documents, often times by PDFs containing the exploit within them. The purpose of it is the enabling of the adversary's further step. Targeting also takes place in the second stage, manifesting when the attacker or it's agent, identify prospective exploitation victims. These two steps are likely to occur, however they are not required.

Following onto the third stage of the first phase, the Cyber Intrusion stage, spanning over Delivery, Exploit and Install/Modify steps. This stage includes the

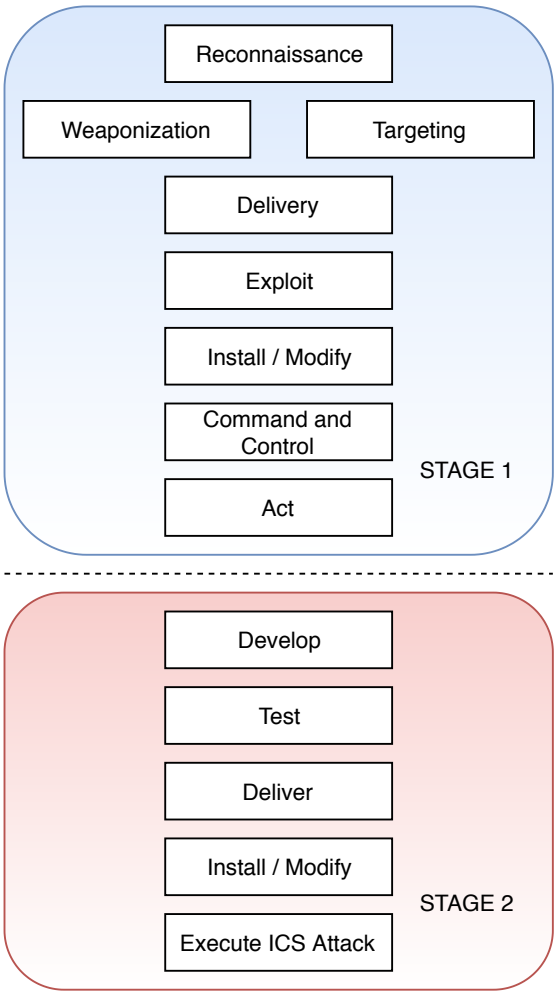

Fig. 1.4: Basic ICS Cyber Kill Chain.

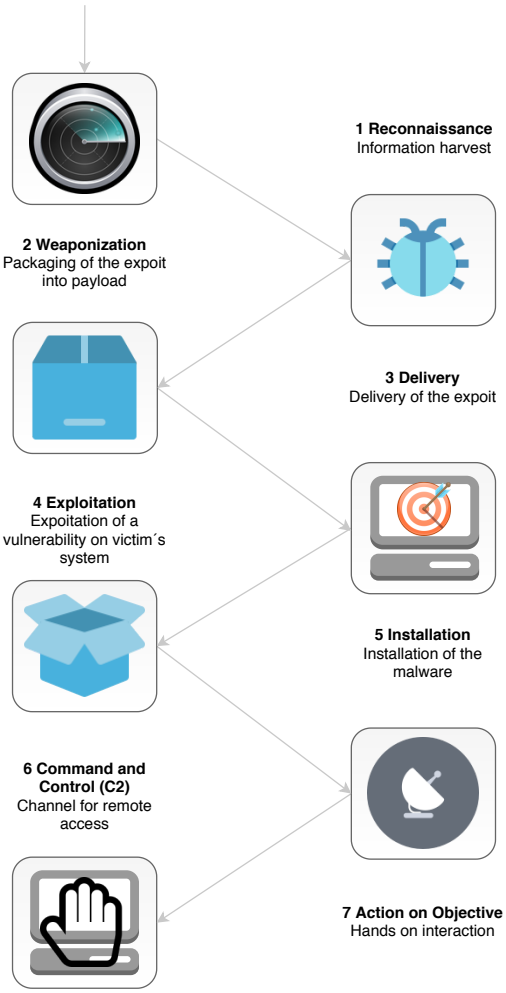

Fig. 1.5: Kill Chain Stage 1.

intrusion step itself either, successful or not, starting with Delivery where interaction with the target's network is initiated, as an example could be a phishing e-mail including the weaponized PDF file. Moving onto the Exploit step, where after the weaponized file is opened, the attacker moves onto performing the malicious actions. Lastly after successful exploitation, installation of remote access malware or modification of systems capabilities follows.

Subsequently, Command and Control step or Management and Enablement step comes to place, in a form of established connection to a previously installed or modified environment. Redundancy of potential paths of connection are favoured to maximize the outcome success rate. Furthermore the attack's connection can be carried out by hiding in the inbound and outbound traffic, inserted in the existing communication or by the implementation of a foreign device that creates attacker's own connection path. After these steps the attacker can move onto the Act step which overlapses to the second stage of the ICS Cyber Kill Chain. The number of

attacks that could be carried out is rather overwhelming, however some will be discussed in following subsections. Moreover, system discovery, installation, capability execution, data collection or trace cleaning are critical phase for a successful follow up stage 2.

Knowledge gained from the steps implemented in stage 1, is vital in further steps of the ICS attack. However, unintentional attacks could be triggered during the stage 1 resulting in outcome of failure. Thus, should be contained during the deployment and the intentional attacks should be carried out in the phase 2. Starting with the Develop step, where the attack is developed and fine tuned for specific impact of the targeted system. This step could take a significant amount of time due to the desire of a successful outcome. This step is rather difficult to detect, by the defender, unless the development is done in live production implying the attacker's ignorance, in favourable case, high level of skill.

Moving onto the Test step or evaluation of the developed capabilities, where the tests are carried out on similar systems to result in a meaningful impact. Varying in their simplicity, the test attacks could be even as simple as denial of service or network scan done in higher volume. The level of impact could be increased if the attackers posses physical ICS hardware or software.

Lastly we move ICS Attack step where the attacker delivers the capability, installs or modifies functionalities and ultimately executes the attack. These attacks vary in their nature and complexity depending on the targeted system, for them to trigger desired conditions required to manipulate the system or process the attacks could be categorized to three categories, Enabling Attacks, Initiating Attacks and Supporting Attacks. Most commonly used methods of ICS attacks are Denial (Denial of View, Denial of Control, Denial of Safety), Loss (Loss of View, Loss of Control) and Manipulation (Manipulation of View, Manipulation of Control, Manipulation of Sensors and Instruments, Manipulation of Safety).

In comparison, IT systems and OT systems operating ICS, have significantly different impact. Whereas IT system exploits mainly impact businesses and data, OT exploits could lead to injuries or casualties of personnel, physical damage to systems or facilities or the decrease of resources being produced by automated systems. Thus the security aspect of ICS systems demands utmost priority [13].

To combat these network security issues, multiple tactics and layers of defense can be implemented in such vulnerable or rather data sensitive and performance critical networks and systems, such as ***Management and Control Plane*** implementation, ***Network Scanning and Enumeration***, ***Lateral Movement and Malware Attacks*** prevention or ***Boundary Controls for Data Protection***. Brief explanation of these tactics and systems follows in the following text.

Management networks and control planes are in use for management, provision-

ing and monitoring of the security systems and networks and components related to them such as firewalls or routers. These types of management networks carry out functions for SCADA and ICS systems that are deemed critical. These networks are exposed to the same threat level as conventional business systems. TAC or Transport Access Control could serve the purpose of authorized access and authentication to the ICS systems and networks, preventing the port scanning, reconnaissance and ultimately making them hidden. Furthermore, authorized as well as unauthorized attempts are being logged by analytic systems or Security Information and Event Management [14].

To ensure the protection against network scanning and enumeration attacks or port scanning, ***Zero Trust Networks*** are being deployed to only respond to requests only if the requesting identity is authorized and authenticated. Thus, unless these conditions are not met on the first sent packet, further interaction does not occur. Protection against SYN scans can redirect unauthorized requests ore traffic to another resource such as dedicated network or to a Honeypot network, a Honeynet, which contains multiple dedicated physical or emulated Honeypot machines.

To combat the lateral movement of malware, a network topology segmentation into network enclaves which are internal network sections, subdivided from the network. Functionality of a network enclave lies in the limitation of access to parts of network. Requiring the authorized identities to either enter or leave. Security measures deployed to prevent lateral movement of the malware also protect the users from phishing type attacks. Furthermore, complimenting the First Attack Authentication, the implementation of TAC provides authentication for the requestor and the requested resource, thus shielding them from the attacks.

Considering the still more and more enforced General Data Protection Regulation (GDPR) and the National Institute of Standards and Technology (NIST) cyber security framework, boundary controls are becoming particularly relevant. Data is mostly filtered from the systems by the network, to combat this issue, remediation controls and autonomic events are being deployed to handle any unauthorized data flow to be either terminated right away or restricted and redirected for threat investigation. This can be done by implementing trust levels into the designated network.

2 Honeypot Systems

This chapter will focus on the introduction of the honeypot systems, the terminology, brief history of honeypots, honeypot functionality based on their characterization and behavioral traits.

Experienced system administrators should be equipped with a handful of tools at their disposal when it comes to preventing, defeating or fending off intruders from their network or work stations. One of these tools that surfaced in the past 20 years is the honeypot systems. More on the history of honeypots will be covered later in this section. A honeypot system differs from traditional intrusion systems or firewalls, that are active systems. The essential purpose of a honeypot system is to deceive perpetrators into thinking that they are interacting with a real network device, by creating emulated systems or by dedicated physical honeypot systems.

A honeypot could be defined as a resource which value is not being attacked or compromised, visible to the attackers and intruders, luring them away from valuable resources. A honeypot is expected to be attacked, probed even fully exploited, a complete opposite of a valuable resource. The reason for this otherwise odd behavior of honeypot is to extract and provide data logs of useful information, which would otherwise be unobtainable or unknown, helping us to learn about the intentions, methods and desires of the attackers and subsequently strengthening our network security systems.

An example of a honeypot infrastructure is shown in Fig. 2.1, where the honeypot sensor is routing packets onto the honeypot systems, which could be an actual physical system or an emulated one. The honeypot sensor could include a intrusion detection system implemented on it, to watch over all incoming and outgoing traffic of this system. These honeypot systems do not store any useful ore valuable information on them, however they are much more attractive to the intruder rather that the production system. The honeypot attractiveness could be achieved by having a running services on them that most intruders would know or consider vulnerable and exploitable, translating into spending time on the honeypot systems rather then the production system. Which could pose as a red flag to the intruder, that the system is actually a honeypot, moreover the goal of these systems is to appear seamlessly integrated into the whole network. The log server stores the log activity sessions, that are used later for further evaluation and research.

The history of honeypots could be traced back to Fred Cohen's ***The Deception Toolkit (DTK)*** publication. That introduced suit of tools that emulated services

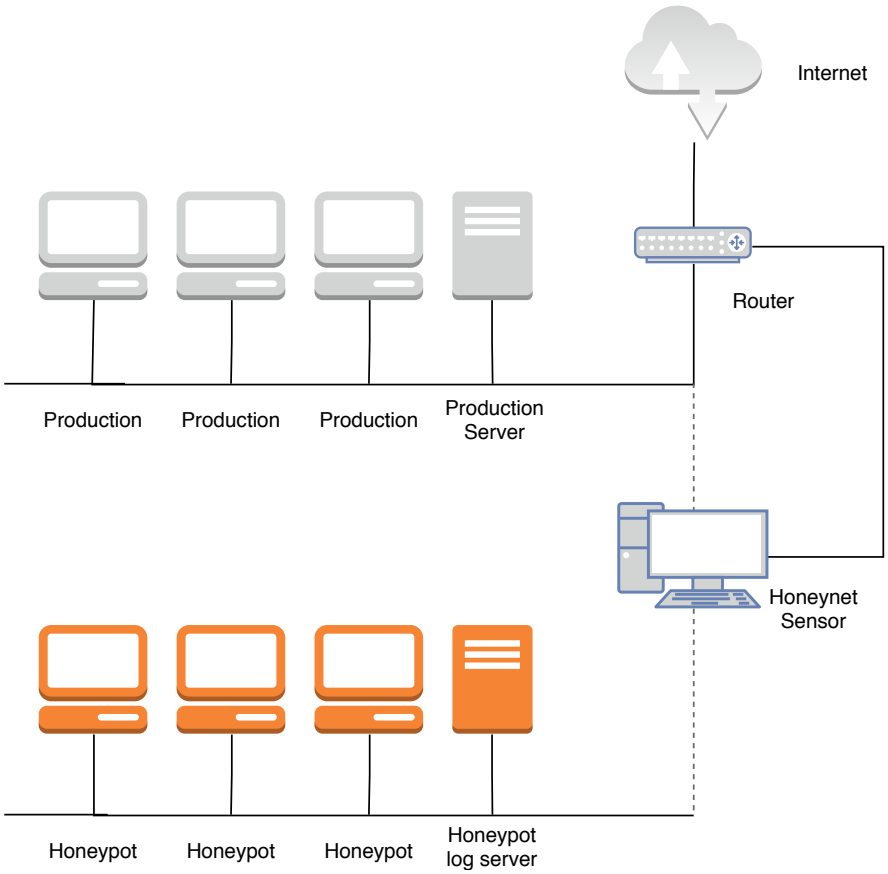

Fig. 2.1: Example of a Honeypot structure.

running on a Unix Systems and were written in PERL and C. This toolkit was used to both detect intrusions as well as to deceive the intruders. This system however emulated the known vulnerabilities of the services, furthermore there was no operating system for the attacker to interact with and subsequently the information that could have been extracted were limited [15].

This does not however, mean that the idea behind a honeypot was not used in earlier iterations, such example could be the ***The Cuckoo's Egg*** or ***"An Evening with Berferd in Which a Cracker Is Lured, Endured, and Studied***, these publications, varying in the technicality yet with similar intent, both dabbled with the idea of what a Honeypot would become in the future [16].

The Deception Toolkit (DTK) was soon followed by the ***CyberCop*** developed by Alfred Huger, differing from the DTK in several features. This system could emulate various systems, such as Cisco router or Solaris server, at the same time. Therefore, the CyberCop managed to emulate network as a whole, with system having their dedicated services to the emulated operating system.

The ***Honeynet*** project was established in 1999 by a group of 30 security researchers, with the intent to research blackhat groups. Their work proved immensely

capable and beneficial in the potential of honeypot systems. ***Know Your Enemy*** included methods of the deployed Honeynets for discovering and learning about the intruders, which only pushed the awareness of the honeypot systems to the spotlight [17].

To this day, a significant amount of Honeypot systems was developed, ranging with their capabilities, behaviors to their method of deployment, more on which will be followed in the next section.

2.1 Honeypots and Intrusion Detection Systems

Standard Intrusion Detection Systems (IDS) deployed on network are able to detect penetration and attack based on the fingerprint of packets or a bit sequence, or by anomaly detection in the network traffic. When it comes to detection by fingerprints, it is fundamental for the Intrusion Detection System to posses the attack fingerprint in it's database, furthermore the type of the attack is known and it has assigned fingerprint. Moreover, the Intrusion Detection System fingerprint database has to be updated. Considering the system is probed or attacked by an unknown type of attack, the detection is worthless. Moving onto the detection of network traffic, there are few parameters that could be spectated and analyzed: data size, ports, source packets/ destination packets or other parameters. This data is then compared to the reference data captured during a casual network traffic. If the system detects an anomaly differing from the reference data, it triggers a warning.

IDS systems share problems considering false anomaly detection, even though no anomaly has occurred. Such case can be made for data backup, while there is no imminent danger, the IDS system evaluates the event as an anomaly for the data flow increased significantly. In contrast, he probability of a honeypot to produce a false case is minimal, for the system itself produces no productive value and no internal connection should occur. Moreover, all connection attempts to the honeypot system is classified as suspicious.

IDS system's functionality is also made even more difficult by the implementation of transferred data security, by implementing security protocols, which subsequently complicate the IDS system flow. Encrypted data, transferred by encryption protocol can not by analyzed by the IDS system and and the attack could be left undetected. Honeypots on the other hand, monitors the system itself not only the network communication.

One of the main advantages of the honeypot systems is the detection of unknown vulnerabilities and attacks (zero-day attacks), by analyzing how intruders managed to penetrate the network to the honeypot and what were their intentions while connected to the honeypot system. If used by the intruder, source codes or tools

can be salvaged and subsequently examined. Another beneficial attribute of the honeypot systems in comparison to IDS systems is the ability to slow down the intruders by spending their time on the honeypot itself. This is all achieved by the Honeypot's ability to simulate number of devices and services, giving the system administrators time to react.

2.2 Honeypot Types

Honeypot systems could be categorized by various specifications, however main factors by which Honeypot systems are distinguished are their **Level of Interaction**, **General Purpose**, **Side of Emulation** and their by **Underlying Hardware**, as shown in Figure 2.2. Level of interaction could be divided to 3 parts, Low-interaction, Medium-interaction and High-interaction. Main differences are associated with the amount of data that can be extracted from these honeypot systems, complexity of the systems such as the services that these system are able to run and provide, level of maintenance required to operate these systems or deployment and resource consumption.

2.2.1 General Purpose

General purpose is rather self explanatory, either used for research or directly in production environment. As illustrated in Figure 2.2, Honeypot systems could be set up to imitate a production environment, as a deception tool to defend networks. This also allows for discovering the vulnerabilities in the production environment, tat would otherwise be left uncheck and could potentially lead to a full exploitation of the system network. The other setup is for research purposes, to explore and evaluate how well and to what extend could a honeypot system mimic a real system, moreover what types of exploits could be used against these types of systems. This type of categorization is not entirely absolute but rather serves as an identification of the honeypot system purpose [18].

2.2.2 Side of Emulation

Another way of classification of the Honeypot systems is by their side of emulation. Honeypots emulated on the server side serve the purpose of emulating network services. In comparison to client side honeypots they lie rather than engaging, they lie dormant waiting for and intrusion engagement. Based on the services that are being simulated on the server side honeypots, the intruder is presented with various approaches in trying to exploit the system. In comparison to server side honeypots,

client side honeypots are significantly more active and aggressive in their nature, for they are searching for servers with suspicious nature. We can assume that all interaction with the server side emulated honeypots is suspicious and most likely hostile, client side honeypots have to distinguish between the nature of servers they interact with. Such a way of determining the nature of the servers interacted with is by statically analyzing and matching signatures on pre-based hostile activity [19].

Server based or server emulated honeypots are historically the most common type of honeypots, considering the side of emulation. Server honeypots function is generally considered passive in their nature, moreover their are laying dormant, waiting for them to be attacked or probed. One of their main advantage is their ability to process high amount of requests, detect computer worms and new network systems exploits. ***Honeyd*** and ***Nepenthes*** are just some of server based honeypots.

Most common targets of malware are however the client stations, considering they store valuable assets, which are targeted by attackers. Client stations are also most frequently targeted systems. Therefore, client based honeypots come to place that simulate basic client behaviors, in a most common form of web browsing. Client based versions also allow to collect malware samples, which are considerably difficult to get by traditional methods. Typical threats such as phishing, web browser vulnerability or other drive by infections. Such client based honeypots are ***HPC Capture*** or ***HoneyMonkey***, that is provided by Microsoft and is used to test software vulnerabilities [20].

2.2.3 Underlying Hardware

One possible differentiation of the Honeypots is between the virtual and physical Honeypots. Physical Honeypot could be described as an Honeypot running on a physical device or machine. Physical Honeypots are often high-interaction, therefore allowing the Honeypots to be completely compromised and exploited. High-interaction is associated with high maintenance, higher price and also difficulty of deployment. Considering there is need for a large system with a vast address space to be deployed, it is highly impractical to deploy this type of Honeypot. This is where virtual Honeypots pose as a viable solution. Virtual Honeypots, compared to the physical Honeypots are much easier to deploy, maintain and most importantly scalable to such an extent that whole network systems or industrial complexes can be simulated from one physical machine. Instead of deploying a physical computer, which sole purpose is to act as a Honeypot, a single physical machine can emulate and host number of virtual machines at once, each emulating different network or industrial system. This can be done by VMware [21] or VM VirtualBox [22], that can emulate several operating systems and applications associated with them, at

the same time, on a single physical machine. For these emulated Honeypots to be reached by the attackers from the Internet and therefore the ability to collect traffic data, virtual machines have to be configured accordingly.

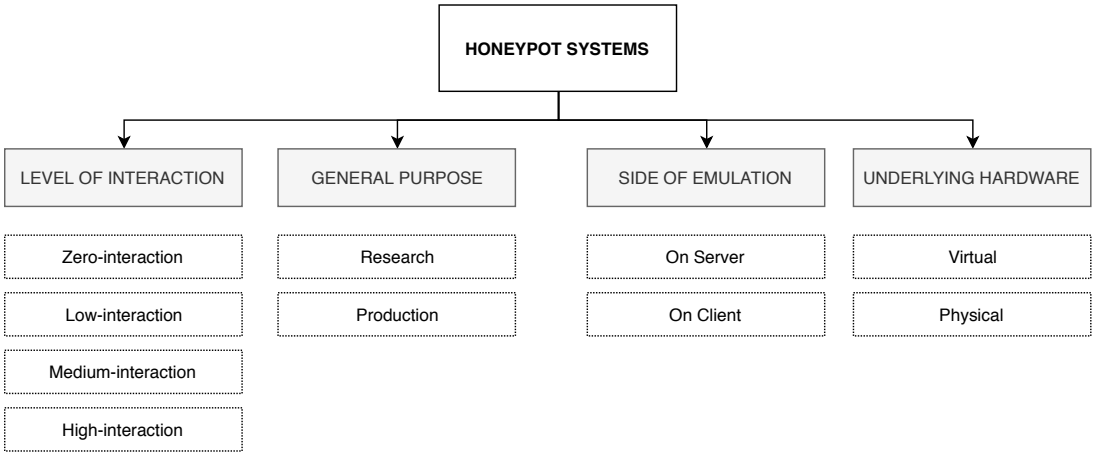

Fig. 2.2: Categorization of Honeypots.

In the following sections honeypot systems will be described in more depth based on their level of interaction.

Fig. 2.3: Interaction legend.

2.2.4 Low-interaction honeypot

Low-interaction honeypots provide only limited and restricted interaction for the attacker or a malware by emulating a limited number of internet protocols, such as IP and TCP, and network services. The Low-interaction honeypot has no operating system for the perpetrator to interact with and acts more so as a static environment, thus the network is not as endangered by these systems. Low-interaction translates into restricted amount of obtainable data and their limited usability in cases such as unauthorized connection detection, spamming activity, worm detection or automated threats.

Main advantages of the Low-interaction honeypots is the ease of deployment, low maintenance, low resource consumption and low complexity of the architecture. On the downside they do provide a limited and inaccurate responses to exploits. This drawback decreases the aiding ability of discovering vulnerability or attack strategies. Honeypots with a low-interaction emulate only few transport layer services of

the operating system such as ports 80 Hypertext Transfer Protocol (HTTP), 135 Remote procedure call (RCP) or 445 for Server Message Blocks over TCP/IP [23]. The Low-interaction honeypot system structure is illustrated in Fig. 2.4. Examples of the Low-interaction honeypots are ***Dionaea*** and ***HoneyD*** [24].

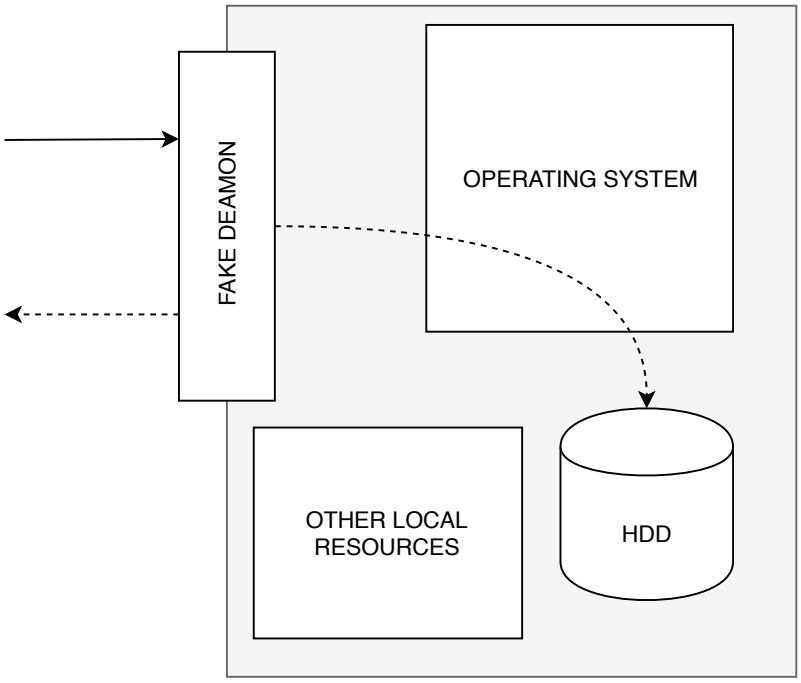

Fig. 2.4: Low-interaction honeypot structure.

2.2.5 Medium-interaction honeypot

Medium-interaction honeypots are capable of supporting slightly more sophisticated emulation of services, still not providing interaction with a operating system, they are also known as ***mixed-interactive honeypots***. Attacker is presented with a more sophisticated illusion of the operating system, thus the complexity of the logged and analyzed data increases.

An example of a Medium-interaction honeypot is presented on a Honeytrap honeypot that creates port listeners dynamically on a TCP based connection that was attempted and extracted from the network stream. Moreover with the increase in complexity, the maintenance and also the possibility of a system compromise increases as well. The Medium-interaction honeypot system structure is illustrated in Fig. 2.5.Examples of a Middle-interaction honeypots are ***Cowrie***, ***Honeytrap***, ***Mwcollect*** and ***Nepenthes*** [25].

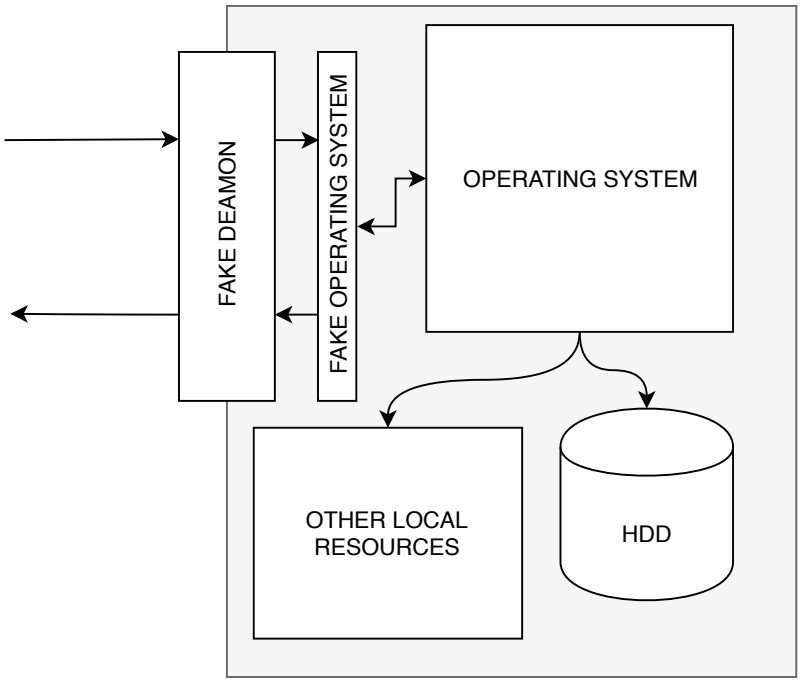

Fig. 2.5: Medium-interaction honeypot structure.

2.2.6 High-interaction honeypot

High-interaction honeypots are complex and most advanced systems that provide a realistic experience for the intruder and gather significantly more useful data for future attack prevention. High-interaction honeypots emulate either fully functional operating systems or it's installation supported by monitoring capabilities. They are most complex, time consuming to implement and high maintenance, also requiring constant surveillance. As mentioned, the perpetrator interacts with a real system, where instead of emulating FTP or a HTTP server, a real FTP or HTTP server is installed which translates to the possibility of gathering sensitive information or a full compromise of the system itself, followed by further sophisticated attacks with even higher possibility of data extraction.

Therefore, because of the high demands of this system, High-interaction honeypots are usually used in a research environment, however they can also be used in a production environment [26]. Their usefulness is however most remarkable in cases of capturing logs of vulnerabilities or weak spots, not yet known to us. They are at their best in so called ***0-Day attack*** situations. The High-interaction honeypot system structure is illustrated in Fig. 2.6. Examples of High-interaction honeypots are ***HonSSH***, ***Sebek*** and ***Honeynets***.

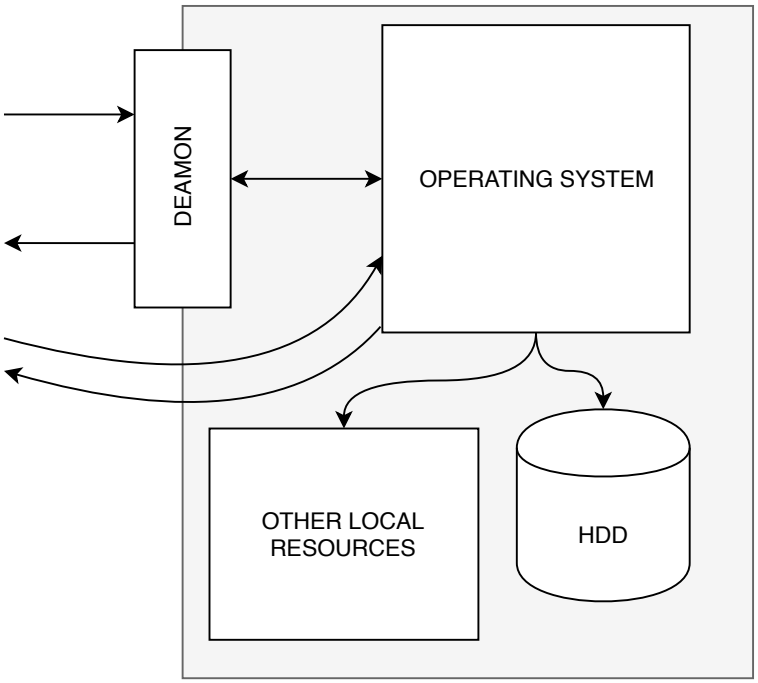

Fig. 2.6: High-interaction honeypot structure.

2.2.7 Zero-interaction honeypot

In 2013 ***"Amber: A zero-interaction honeypot and network enforcer with modular intelligence"*** was published, introducing a so called ***Zero-interaction honeypot*** presenting a honeypot that provided no emulated services and the function of these honeypots rather then harvesting information for later evaluation is to obstruct the probing sequence of the attacks and make it significantly difficult by using Decision through Presence (DtP) approach to log and produce a black-list of intruding groups later to be passed to the network administrators. The Amber project has proven itself, in the defending of networks, to be useful alternative as well as low-cost solution [27].

Zero-interaction honeypot could also be compared to a system called a ***network telescope***, also known as packet telescope. Network telescope function is based on the principle of monitoring unused (dark) address space of networks, for all activity and traffic linked to these address spaces is categorized as suspicious. Moreover, by default there are no client stations connected to these address spaces, therefore the probability of spotting a probe of these address spaces is much higher with the assumption that the attempt has malicious intent [28].

2.2.8 Honeytokens

Honeytokens could be described as simple honeypots, that do not emulate fake network devices or services. The however emulate fake data that could otherwise be interpreted as sensitive and valuable, such as database entries, user accounts with personal information, IP addresses or Uniform Resource Indicators (URL). Their primary function is to detect intrusion entries without any behavioral interference. Generally occurring as an attractively named file or repository, from the perpetrator's point of view. Honeytoken could reside on a production system as a detection tool, implemented into server repositories or databases.

A way of detecting that someone is trying to access these files or repositories, could be managed by creating a intrusion detection system (IDS) signature, that searches for packets containing the key-phrases within the targeted documents. With egress filtering, a way of monitoring packets exiting from the inside the network to the outside of network, if the set of rules is violated it indicates that someone tried to access the document. Another way of detecting these behaviors is through the access time-stamp, which updates each time the document is accessed, copied or altered in any way. Honeytokens once again do not protect against the alternation of the documents, instead further the network administrators knowledge of the attacker's behavior. [29].

2.3 Honeypot implementations

This section will focus on the theoretical aspects of selected list of Honeypots, their basic attributes, brief history, current stale of development, availability, supported protocols, distribution, advantages and disadvantages. These sections will not however include the initial installation, configuration and subsequent deployment of the Honeypots. This will be discussed in detail in the Honeypot deployment chapter.

2.3.1 Pentbox

To understand the concept of Honeypot system deployment, ***PenTBox*** will be the first on this list, for it's ease of use and versatility. PenTBox is a Security Suite that packs security and stability testing oriented tools for networks and systems. Programmed in Ruby and oriented to GNU/Linux systems, but compatible with Windows, MacOS and every systems where Ruby works.

PenTBox is an open-source security suite. This software provides user with probing, stability and security testing tools for systems and networks. It is targeted primarily to GNU/Linux systems, however it's compatibility spreads over all systems that support Ruby, in which is the PenTBox programmed.

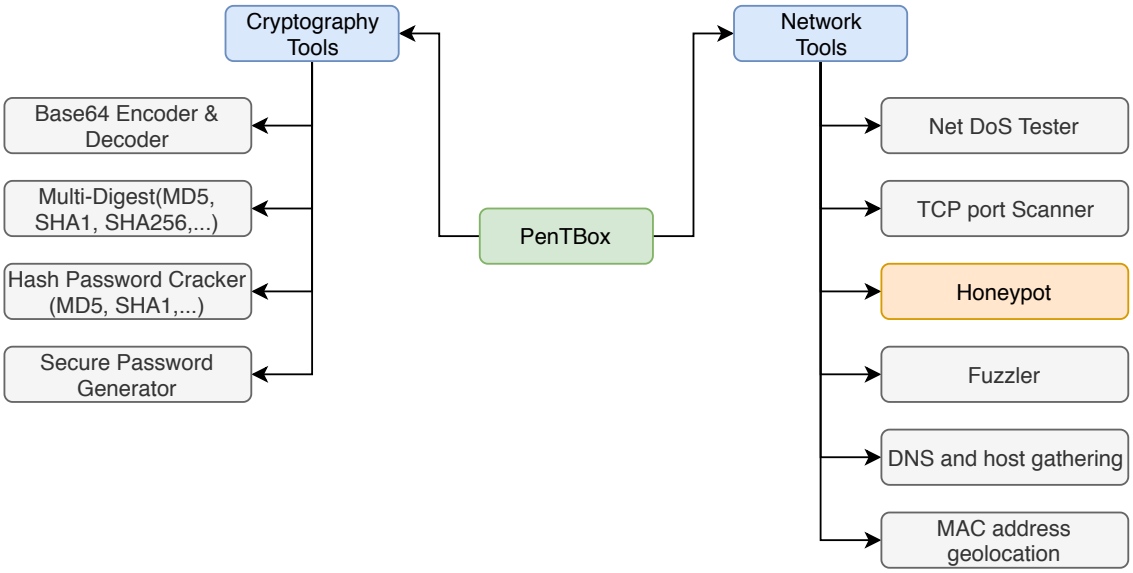

Fig. 2.7: PenTBox capability set.

Installation, deployment and captured logs of the PenTBox suite will be described in the Deployment of Pentbox section.

2.3.2 Cowrie

Cowrie Honeypot is a forked Kippo project, which is no longer under development. Cowrie falls under the medium to high interaction Honeypot branch. It is a SSH and Telnet Honeypot developed to log brute force attacks and the shell command interaction carried out by the perpetrator.

In it's medium interaction state, shell state, it emulates a UNIX system in Python. In it's high interaction mode, proxy state, it acts as an SSH and telnet proxy to over-watch the behavior of the attacker to another, targeted system.

- ***Emulated shell (default state)***
  Forged filesystem with the ability to add files and to remove them. A full forged filesystem bearing a resemblance to a Debian 5.0 installation is included.
  Ability of planting forged file contents so the attacker can `cat` files such as `/etc/passwd`. Bare minimal of file contents are included.
  Cowrie stores files downloaded with `wget` or `curl` commands or uploaded with SFTP and SCP protocols for a later inspection.

- ***SSH and telnet proxy***
  Pure telnet and ssh proxy with the commands monitoring.

Or let Cowrie manage a pool of Qemu emulated servers to provide the systems to login to

- ***For both states***
  Storage of session logs in UML Compatible format for replay with the `bin/playlog` utility.
  File upload with support of SFTP and SCP.
  SSH exec commands support.
  SSh proxying, direct-TCP connection attempts logging.
  SMTP connections forwarding to SMTP Honeypot (mailhoney).
  JSON logging for post-processing in log management utilities.

As mentioned above, Cowrie emulates a UNIX system, more specifically a system reminiscent of a Debian 5, in which files can be added or potentially removed. Cowrie saves files, that are downloaded via `wget` or `curl`, eventually uploaded through Secure File Transfer Protocol (SFTP) or Session control protocol (SCP). Cowrie also allows for Simple Mail Transfer Protocol (SMTP) connection to a SMTP Honeypot. Commands `wget` and `curl` use protocols such as HTTP, HTTPS, FTP, FTPS, etc.

- ***SSH*** - Secure Shell a TCP cryptographic network protocol populating port 22, used for secure service operation through an unsecured network system. Used primarily to access UNIX based systems. Developed as a direct successor to the Telnet protocol and other unsecured remote shell protocols such as Berkeley rlogin, rsh, or rexec. Providing the data integrity and confidentiality over unsecured network systems.

- ***Telnet*** - Client server TCP protocol, populating port 23, basing it's functionality on reliable connection oriented transport. Telnet provides bidirectional text oriented communication. However, due to serious security concerns considering the communication over unsecured networks, the use of SSH overshadowed the Telnet protocol.

2.3.3 Conpot

Conpot is a low-interaction Industrial Control System Honeypot. This server side Honeypot was designed with the ease of use, deployment, modification and extension in mind. Conpot provides range of Industrial Control System protocols enabling the network administrators to emulate vast range of complex infrastructures to deceive the intruders into thinking they have stumbled upon an industrial facility.

By adjusting the configuration files the system can emulate service response

times that are feigned to act as they were transmitted by a real device that is under continuous traffic load. This behavior adds to the deception capability. Conpot also allows for a customized Human Machine Interface (HMI) to increase the attack surface of the Honeypot. Furthermore, Conpot provides full stack of protocols, allowing the Honeypot to be extended by a physical device or to be accessed with a productive Human Machine Interface. The Conpot Honeypot is developed by the Honeynet Project in association with other companies.

With it's first release dating to the May 2013, the project has undergone significant improvements and is still under an active development and support of the community. Conpot is categorized as object-oriented low-interaction implementation of a Industrial Control System (ICS)/ Supervisory Control Data Acquisition Systems (SCADA) Honeypot, programmed in Python. Description of ICS and SCADA systems was mentioned in earlier sections. Protocols that are supported by the Conpot Honeypot, with their emulated servers, are:

- ***Modbus*** - on Modbus server, populating the 502 port, this protocol is explained in the Communication protocols in OT section.

- ***S7comm*** - on S7 server, a proprietary Siemens protocol that communicates between Programmable Logic Controllers (PLC) of the Siemens S7 devices on the Application Layer, Presentation Layer and Session Layer of the OSI model. It's use is in PLC programming, PLC's data exchange, PLC data access from SCADA systems and diagnostics.

- ***Kamstrup*** - Kamstrup meter on Kamstrup server and Kamstrup management on Kamstrup management server. Populating port 601, Kamstrup is a serial request response protocol, using the Physical Layer, Data link Layer and Application Layer.

- ***HTTP*** - on HTTP server, populating port 80. Hypertext Transfer Protocol is a request response, client server relation application protocol for collaborative, distributed and hypermedia informational systems. It's secure version is HTTPS which populates port 443.

- ***SNMP*** - on SNMP server, Simple Network Management Protocol (SNMP) is an Application Layer protocol using UDP populating port 161, exchanging management information between devices of the network. Moreover, it is used to manage and monitor the elements of the network.

- ***IPMI*** - on IPMI server, Intelligent Platform Management Interface is a network protocol populating port 623 using UDP, sometimes TCP. It's purpose is to manage a potentially powered off computer or unresponsive, using a network connection to it's hardware instead of connecting to the to an operating system or shell login. Another use could be in remote installation of a custom operating system.

- ***ENIP*** - on ENIP server, EtherNet/IP (Industrial Protocol), serves the purpose of adapting the Common Industrial Protocol to standard Ethernet. Explicit messaging is done by the TCP port 44818 and implicit messaging by the UDP on port 2222. It' use is widely spread in factory and process industries.

- ***IEC104*** - on IEC104 server, is a part of IEC Telecontrol Equipment and Systems Standard IEC 60870-5, a client server model providing communication profiles for telecontrol messages in the middle of two systems in the power system automation and electrical engineering, populating the TCP port 2404.

- ***FTP*** - on FTP server, a client server model protocol, populating TCP port 21. It's functionality is in computer file transportation between server side and client side, either in active or passive state which determines the data connection establishment.

- ***TFTP*** - on TFTP server, Trivial File Transfer Protocol is a client server model, simple fault tolerant File Transfer Protocol (FTP) with wide use in the initial stages of network and node booting, widely used for it's ease of implementation. Using UDP on port 69.

2.3.4 MiniCPS

Is a security research toolkit for Industrial Control System security. This platform is built on top of the Mininet, which is a lightweight Linux network emulation platform, MiniCPS extends the Mininet application a capabilities to the domain of Industrial Control Systems. MiniCPS compacts and facilitates several partitions starting with network emulation, Industrial Control System devices simulation to

which physical processes emulation can be attached, ultimately building a real-time Industrial Control System simulation all in one. This framework is written in Python, providing object oriented, high level public API, free and open source under the MIT license. Fig. 2.8 illustrates the MiniCPS framework layers and how these parts are connected by the `Network Emulation`, which is done by utilizing MiniNet, while the `Component´s Logic` communicate with the `Physical Layer Simulation`, throught the `Physical Layer API` (Application programming interface).

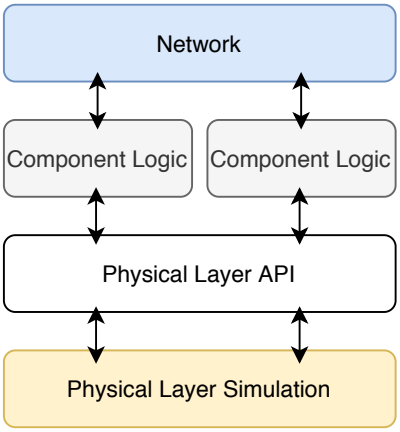

Fig. 2.8: MiniCPS framework layers

This work attempts to implement the MiniCPS framework to act as a fully capable High-interaction ICS Honeypot with respected additions, capable of emulating simple devices utilizing Modbus and ENIP (EtherNet/IP) protocols, while generating a master-slave communication for the respected ICS protocols, which runs on an emulated MiniNet topology. This topology can can emulate PLCs, middleboxes, Human Machine Interface (HMI) with dedicated MAC addresses, IP addresses and net masks. When the attackers discover the topology, they are presented with this data, and can interact by planting their own packets. Moreover, the attackers are further persuaded by the MiniCPS´s link shaping capabilities, which translates into custom latency, custom link bandwidth or packet loss [30].

The MiniCPS platform was chosen as a liable solution to the development of the new High-interaction Honeypot, for it´s previous implementation in the ***Towards High-Interaction Virtual ICS Honeypots-in-a-Box*** [31]. Where the MiniCPS framework served as a water treatment testbed imitation and is considered one of first academically developed High-interaction Industrial Control System Honeypot solution, with the ability to implement a Software-Defined Network (SDN) controller management. This solution was deployed without utilizing a fully virtualized environment.

Furthermore it´s capabilities were tested in a CFT (Capture-The-Flag) competition hosted by the Singapore University of Technology and Design (SUTD) in July 2016, as a part of an Industrial Control System event, bearing the name ***SWaT Security Showdown (S3)***.

2.3.5 MiniNet

As previously stated MiniCPS builds on top of a light weight Linux emulation platform called MiniNet. The MiniNet platform allows for emulation of realistic virtual network topologies, that can facilitate number of hosts, routers, switches and link all running on a solitary Linux kernel [32]. By utilizing virtualization of a light weight nature, it has the ability to act as a complete network, running same system, kernel and user code on a single system. It´s use spans over teaching, through research to development. Not only for it´s simple interaction through Command-Line Interface (CLI), customization options, real hardware deployment, it´s open sourced nature facilitated under the BSD Open Source license, but also it´s support of OpenFlow (for customize packet forwarding) and Software Defined Networking (SDN), the MiniNet is considered as vital and rather useful tool, well suited for this thesis goals [33]. Another useful facts is that is under a constant development and support from the community. As previously mentioned, MiniNet allows the Software Defined Networking. This solidifies the usability in this thesis implementation, for there is no need for real hardware to be present. Furthermore, MiniNet networks can run real code, real Linux kernel, network stack and also Linux/Unix network applications. Mininet also provides Python Application Programming Interface of extensive nature, allowing for experiments to take place and also network emulation.

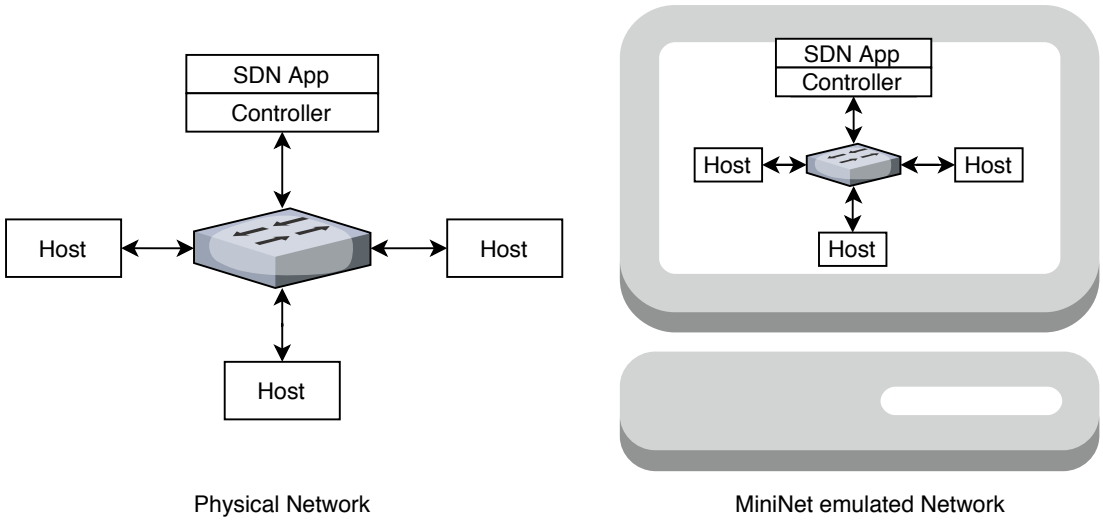

Fig. 2.9: MiniNet virtualization.

MiniNet network can consist of:

- ***Isolated Hosts*** - Group of processes running on the user level, transferred into a namespace which provides ownership of the designated interfaces, routing tables and ports.
- ***Emulated Links*** - Data rates of every link are enforced by the Linux Traffic Control to configure the rate accordingly to the developer´s needs. Virtual Ethernet interface of interafeces are assigned to each and every one of the emulated hosts.
- ***Emulated Switches*** - Packets are moved across the interfaces by utilizing either the Open vSwitch activated in kernel mode or the default Linux bridge. Routers and switches can be used in either the user space or the kernel.

2.3.6 HonSSH

HonSSH is considered as a High-interaction Honeypot solution, even though it´s usage is as a proxy attached to a High-interaction Honeypot and it is designed to work in conjunction with it. This section part focuses on the main functionality description of the HonSSH Honeypot solution, some of the main function from the main branch will be explained further bellow in this section. Main functionality of the HonSSH Honeypot is to assume position between the attacker and the attacker´s target, a Honeypot in this case, and to create two separated SSH connections. To the attacker, this appears as a direct SSH connection however. All connection associated with the HonSSH is logged and preserved in several log files, with the ability to send these logs to email as an alert, these logs include failed connection attempts as well. One of the main functionalities of the HonSSH is password spoofing, which can be configured in several ways, however it´s main functionality is to replace the password typed by the attacker with the real password of the Honeypot, assuming that the guessed password is located in the configuration file. Assuming that the selected passwords are considered as weak (root, toor, admin, 123456). HonSSH is a forked project from the Kippo Honeypot solution, inheriting some of it´s functionality and building upon it. Therefore, thanks to Kippo, all interaction is captured and logged into a `TTY` logs, with the ability to replay the whole interaction session by utilizing the `playlog` utility inherited from the Kippo Honeypot. Another functionality inherited from the Kippo Honeypot is the ability to step into, or rather hijack the session in real time, by utilizing the Telnet management interface, this however may only be used if the intrusion is spotted in time. Furthermore, the attacker´s sessions are captured in a text based summaries. When the attacker connects to the honeypot, they are able to utilize the `wget` and `Secure copy protocol (SCP)` to download and copy files into the Honeypot environment. HonSSH can be configured as either a

static Honeypot of with the Docker engine, which allows it to reuse Honeypots in the IP basis, furthermore save modifications done to the Docker container by utiliying the filesystem watcher. Another of the main branch features of the HonSSH is Advanced Networking which will be described in further detail further in this section. Last but not least, HonSSH allows for integration of Honeypot facilitator´s output scripts. Fig. 2.10, shows the visualization of the HonSSH Honeypot functionality [34].

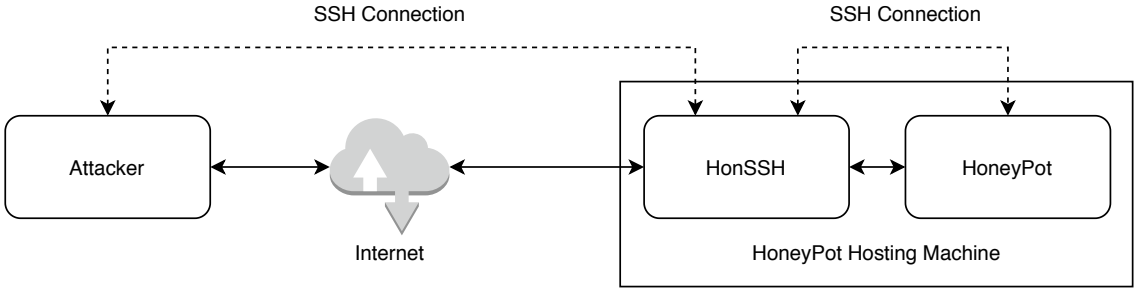

Fig. 2.10: HonSSH functionality

Upon the first initialization of HonSSH a new connection is established with the Honeypot, acquiring version of the SSH server running on the Honeypot, storing the information and subsequently disconnecting from it. Upon the connection initiated from the attacker´s side, connecting to the HonSSH, HonSSH initiates and separates and forwards the connection from the attacker to the Honeypot. After a successful connection all transferred data is forwarded from the attacker to the Honeypot, this process is bidirectional. The intentions of the creator was that the HonSSH provides both firewall blocking and NAT translation. As previously mentioned HonSSH provides additional functionalities of the main branch, one of which is Password Spoofing. The purpose of the password spoofing capability is to substitute the wrong password provided by the attacker, by the correct password, allowing them to subsequently log in to the Honeypot successfully. HonSSH supports multiple attackers passwords to be spoofed at the same time and provides two option of how the password should be spoofed: statically or randomly. While ***Static/Fixed***, a list of passwords that is assumed to be guessed by the attacker is stored in a field in the configuration file. ***Random***, a random percentage value is inserted in the configuration file, which will allow the inserted percentage of any password guess by the attacker.

Listing 2.1: Static password spoofing.

```
[root]                                                          1
real_password = realpassword                                    2
fake_passwords = fakepassword1, fakepassword2,                  3
   fakepassword3,...
```

This makes the system to accept any string of the `fake_passwords` field.

Listing 2.2: Random password spoofing.

```
[root]                                                          1
real_password = realpassword                                    2
random_chance = percentage probability [0-100]                  3
```

The `percentage probability [0-100]` defines the probability of acceptance of the inserted password, even if the password inserted is not the `realpassword` or any inserted string. To enable the password spoofing, the `users.cfg` file, in which the fake passwords or the percentage of acceptance is stored, is accessed by the `honssh.cfg` in which the password spoofing must be enabled by setting it to `true`. When someone logs in using a spoof password, the details are stored in logs/spoof.log by default. HonSSH provides logs in which it can correlate IP addresses based on the same password used to log into the honeypot. Another of the main functionalities is the Advanced Networking. If it stays in the preconfigured state as disabled, the followed scenario takes place, illustrated in the Fig. 2.11.

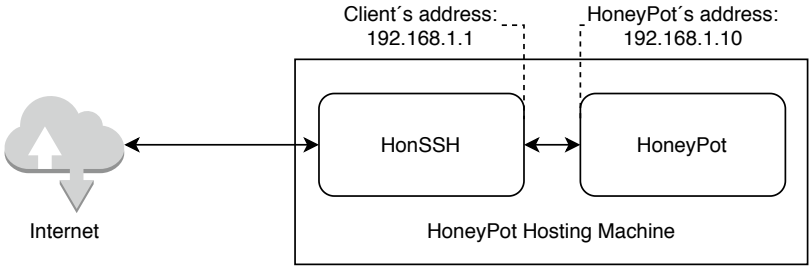

Fig. 2.11: Advanced Networking disabled

What this translates into is that the Honeypot always perceives the incoming connection as if it was coming from the 192.168.1.1 (Client´s address), not the attacker´s IP. This might manifest into unwanted realization of the attacker that they are connected to a Honeypot. By enabling the Advanced Networking feature in the `honssh.cfg` file, HonSSH is allowed to create fake IP addresses, by using `ip link` and `ip addr` commands, therefore masquerading as if all the packets are coming from the attacker. Moreover, root privileges are required to run these commands. if this requirement is not met, HonSSH falls back into it´s default state.

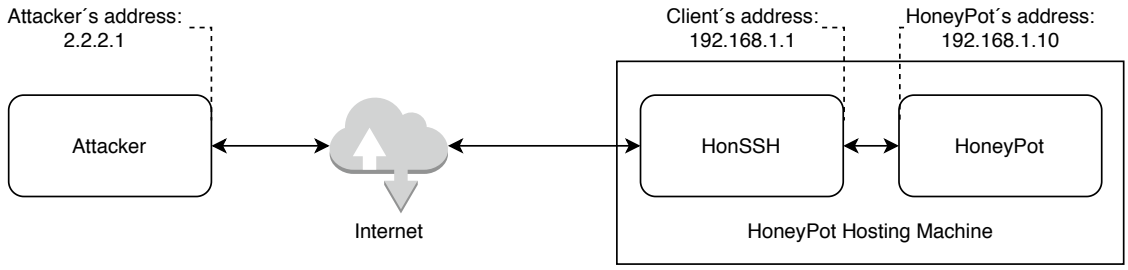

Fig. 2.12: Advanced Networking enabled

When the connection is established by the attacker from 2.2.2.1, HonSSH initiates a dummy interface bearing the name `honssh`.

Listing 2.3: HonSSH dummy interface.

```
root@template:#ip link add name honssh type dummy
root@template:#ip link set honssh up
```

After creating the interface it assigns it a semi-random IP address. HonSSH will bind to this address to create the tunnel between HonSSH and the honey pot.

Listing 2.4: HonSSH semi-random address bind.

```
root@template:#ip addr add 2.2.2.2/32 dev honssh
```

This bounds HonSSH to a semi-random IP address which is fake, moreover there are few `iptables` NAT rules to translate the connection between the fake semi-random address and the attacker's IP address.

Listing 2.5: Translation of the source address.

```
root@template:#iptables -t nat -A POSTROUTING -s 2.2.2.2/32
    -d 192.168.1.10/32 -p tcp --dport 22 -j SNAT --to
   2.2.2.1
```

Source address of the packets is translated, leaving the fake IP address for the Honeypot acting as is the packets were originating from the attacker.

Listing 2.6: Translation of the packet destination.

```
root@template:#iptables -t nat -A PREROUTING -s
   192.168.1.10/32 -d 2.2.2.1/32 -p tcp --sport 22 -j DNAT
   --to 2.2.2.2
```

Packets assume the appearance, as if they were initiated by the attacker´s side. Upon the end of the session, HonSSH checks for other opened sessions, if none are detected, the fake IP is removed along with the iptable rules.

Listing 2.7: Fake IP removal with the iptable rules.

```
root@template:#ip addr del 2.2.2.2/32 dev honssh                    1
root@template:#iptables -t nat -D POSTROUTING -s 2.2.2.2/32         2
    -d 192.168.1.10/32 -p tcp --dport 22 -j SNAT --to
   2.2.2.1
root@template:#iptables -t nat -D PREROUTING -s                     3
   192.168.1.10/32 -d 2.2.2.1/32 -p tcp --sport 22 -j DNAT
   --to 2.2.2.2
```

If the HonSSH does not have any opened sessions, the HonSSH interface will be removed as well [35].

2.3.7 Suricata

Suricata is a free Intrusion Detection System (IDS), of open source nature, Open Information Security Foundation (OISF), with sturdy and fast network detection engine for threats, with the option to work as Intrusion Prevention System (IPS) as well. As mentioned it´s able to function as a stand alone Intrusion Detection System, Intrusion Detection System, Network Security Monitoring system (NMS) and alsowork in offline state while processing `.pcap` files. Suricata utilizes rule sets with signature language while inspecting the traffic on the network, furthermore a lightweight, high-level, multi-paradigm programming language `Lua` is used to support complex threat detection. Integrating `JSON` and `YAML` log formats are interpretable by tools such as Grafana, Splunk, Kibana, Logstash/ElasticSearch, etc. [36].

2.3.8 Grafana

Grafana was chosen as the visualizing software solution, as it is an open source observability platform for monitoring applications, infrastructure status, ranging as far as to power plants. It´s features span from visualization through alert options to a vast range of database imports. Visualization can be adjusted with numerous options, allowing for a personalized visualization. It allows to create reusable and dynamic dashboards based on templates, while implementing queries and various data sources. It allows for implementing extensions from plugin libraries, based on the needs of the developer. Considering this thesis, the Grafan server will be running on the hosting machine on the port 3000, and upon firewall configuration, it can be accessible from any machine, provided the login and password are known to the accessing party

3 Honeypot deployment

In this chapter, the deployment of various Honeypot systems will be described and evaluated. Starting with the installation process of the system, following onto the configuration of the Honeypot system to specify what ports and what services should be opened and over-watched, concluding with the captured data from each Honeypot log-file system or third party capture software. Each of these systems will be evaluated based on their ability to be configured and what data their are able to provide to the system administrator. Whole environment will be emulated with VM Virtualbox developed by Oracle.

Each environment will have it's own dedicated Virtualbox image, for there were numerous events, in the early stages of implementation of the Honeypot systems on the virtual environment, where different partitions required to run each system had interfered with one another. To engage the Honeypot systems, a 64-bit Debian (Kali) virtual machine was created. This virtual machine will not store any applications and it's sole purpose will be in accessing the other virtual machines, for that reason minimal resources were dedicated to this virtual image.

Each Honeypot system will be deployed on separate virtual machine to avoid partition collisions. This decision was made after several issues during the deployment of multiple Honeypot systems onto one virtual machine. Ubuntu, a free and open source Linux distribution was chosen for housing of the Honeypot systems, for the selected Honeypot systems are supported by this environment. Each of these virtual images are in a NAT Network, with access to the internet and accessibility to one another.

With a healthy dose of scepticism we assumed that the Honeypots systems that will be deployed, do not include all the necessary tools and extensions to capture every form of attacks or probe attempt. Moreover, based on this assumption additional tools were used during the tests such as Wireshark (under development of the Wireshark team), Snort (developed by Cisco) and Suricata (developed by the Open Information Security Foundation), each with their respective capability set.

3.1 Deployment of Pentbox

As was previously mentioned in the Pentbox Section, PenTBox is a Security Suite that includes stability and security tools. Programmed in Ruby, oriented on the GNU/Linux systems, yet compatible with Windows, MacOS and every other systems supporting Ruby.

PentBox installation is rather straight forward, for the installation is done on a clean image of Linux (Ubuntu), without any additional partitions previously installed, the installation process is following.

Update the packages index:

```
ubuntu@ubuntu-VirtualBox:~$sudo apt update
```

Ruby and wget installation:

```
ubuntu@ubuntu-VirtualBox:~$sudo apt install ruby-full wget
```

Potentially check installed Ruby version:

```
ubuntu@ubuntu-VirtualBox:~$ruby --version
```

After successful Ruby and wget installation, we download a tar file from a source:

```
ubuntu@ubuntu-VirtualBox:~$wget http://downloads.
   sourceforge.net/project/pentbox18realised/pentbox-1.8.
   tar.gz
```

After the zipped file is downloaded we unpack it:

```
ubuntu@ubuntu-VirtualBox:~$tar -zxvf pentbox-1.8.tar.gz
```

After successful unpack of the files we enter the newly created directory where the pentbox.rb is stored and we initialize the PenTBox with the following command:

Listing 3.1: Pentbox initialization.

```
ubuntu@ubuntu-VirtualBox:~/pentbox-1.8$ sudo ruby pentbox.
   rb
```

After executing the command, we are presented with a simple User Interface which is navigated by number insertion. PenTBox presents number of choices, but for the purpose of this thesis only one tab will be used. As shown in Fig. 2.7 Honeypot can be found under the Network Tools tab. After triggering the Honeypot we are presented with two options: `1-Fast Auto Configuration` and `2-Manual`

`Configuration [Advanced Users, more options]`. If Auto Configuration is selected, PenTBox opens and starts to listen on port 80 (HTTP) with a following message:

Listing 3.2: Honeypot activation.

```
HONEYPOT ACTIVATED ON PORT 80 (2019-11-27 21:50:23 +0100)
```

Following this message, we can assume that the PenTBox Honeypot has been started and is ready to capture the opened port requests. To test the functionality of the Honeypot, we engage the port 80 (HTTP) with internet browser, by executing the virtual machine IP address request. The Honeypot detects this connection attempt and reports it following log shown in Listing 3.3.

Listing 3.3: Pentbox HTTP intrusion.

```
INTRUSION ATTEMPT DETECTED! from 10.0.2.8:37112
   (2019-11-27 21:44:56 +0100)
------------------------------
GET / HTTP/1.1
Host: 10.0.2.8
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
   :70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q
   =0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

This log provides us with following information of the attacker (in this case the host machine of the Honeypot).

Listing 3.4: Pentbox HTTP intrusion, part 1.

```
INTRUSION ATTEMPT DETECTED! from 10.0.2.8:37112 (2019-11-27
    21:44:56 +0100)
```

First line informs about basic information of the attacker, what IP address was the attack originated from, on which port, the date and time of the attack and lastly the timezone.

Listing 3.5: Pentbox HTTP intrusion, part 2.

```
GET / HTTP/1.1
Host: 10.0.2.8
```

Following onto the line 3, a HEAD request with three variables enclosed: Request Type, Document URI, and Protocol with the Request Type: GET, Document URI: contains forward slash (/) which is a request for the domain root index file and Protocol: HTTP/1.1. Line 4 shows the IP address of the attacker's device.

Listing 3.6: Pentbox HTTP intrusion, part 3.

```
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv      1
   :70.0) Gecko/20100101 Firefox/70.0
```

Line 5 contains User-Agent request header containing a specific string that enables the network protocol peers to recognize the application type, operating system, software vendor/version of the requested software user agent. Syntax is as followed, `User-Agent: <product> / <product-version> <comment>`, in this case the request was sent from a Mozilla internet browser running on Ubuntu OS.

Listing 3.7: Pentbox HTTP intrusion, part 4.

```
Accept: text/html,application/xhtml+xml,application/xml;q   1
   =0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5                             2
Accept-Encoding: gzip, deflate                              3
Connection: keep-alive                                      4
Upgrade-Insecure-Requests: 1                                5
```

Moving onto the line 6, we are presented with default Accept values, these values are sent when the context does not provide further information, followed by factor weighting `;q=`. This type of message is usually used for requests initiated from the address bar of a internet browser. Line 7 informs about the `Accept-Language:` header, which advertises what languages is the client able to comprehend, and preferred locale variant, followed by factor weighting `;q=`. Line 8 includes the Accept-Encoding request that advertises encoding of the content, most likely a compression algorithm, that is understood by the client. Directives are `gzip` a compression format using the Lempel-Ziv coding (LZ77), with 32-bit cyclic redundancy check and `deflate`, compression format utilizing the zlib structure, with deflate compression algorithm. Moving onto the line 9, the Connection header which controls whether the network connection should stay open after the end of current transaction or not. In this case, the value is keep-alive so the connection persists to be opened, allowing for follow-up requests to the server. Finishing with the line 10, `Upgrade-Insecure-Requests:` signals to the server the client's preference for an authenticated and encrypted response.

To test the `2-Manual Configuration [Advanced Users, more options]` capability, various ports were open sequentially to get a better understanding of what logs are generated based on the opened ports. This process is rather straight forward. Starting with what port the user wants to be opened, what message should be generated upon the connection to the Honeypot, whether to save the logged event and the path and directory of the file, with either default of custom path and finally whether the user should be notified by a sound, if an intrusion is detected. The process is shown in following Listing 3.8.

Listing 3.8: Pentbox Honeypot manual configuration.

```
->2                                                                    1
Insert port to Open.                                                   2
->22                                                                   3
Insert false message to show.                                          4
->established                                                          5
Save a log with intrusions?                                            6
(y/n)->y                                                               7
Log file name? (incremental)                                           8
Default:*/pentbox/other/log_honeypot.txt                               9
->pentbox_ssh.txt                                                      10
Activate beep() sound when intrusion?                                  11
(y/n)->n                                                               12
HONEYPOT ACTIVATED ON PORT 22 (2019-11-25 19:33:21 +0100)              13
```

We are presented with message indicating that the Honeypot opened the port 22 for SSH connection. We engage this port from Kali virtual machine. PenTBox was able to detect the intrusion, notifying us with a following message:

Listing 3.9: SSH log trigger.

```
INTRUSION ATTEMPT DETECTED! from 10.0.2.4:38698 (2019-11-18    1
    11:26:13 +0100)
------------------------------                                 2
SSH-2.0-OpenSSH_8.0p1 Debian-4                                 3
```

This log provided us with initial information about the attacker, same as with the HTTP log. On the Debian (Kali) side, after initiating the SSH connection to the PenTBox, we were presented with the following message:
`kex_exchange_identification connection closed by remote host`
without the message that was specified during the Honeypot configuration which was supposed to be received after successful connection, if we run the command again with a `-v` (verbose) parameter, the message is shown. The PenTBox Secure

Shell (SSH) log provides only with the version on the SSH protocol and the OS of the attacker.

Following the SSH test was the `nmap` probe. Multiple terminals running PenTBox Honeypots were up during this probe, to see if all were visible. After triggering the `nmap`, list of opened ports was show, including all the ports that were opened by PenTBox. However, none of the running terminals captured this probe attack neither any log was acquired.

Listing 3.10: Nmap initiation from Kali host.

```
root@kali:~# nmap 10.0.2.8                                          1
Starting Nmap 7.80(https://nmap.org) at 2019-11-30 09:25            2
   GMT
Nmap scan report for 10.0.2.8 (10.0.2.8)                            3
Host is up (0.00015s latency).                                      4
Not shown: 999 closed ports                                         5
PORT    STATE SERVICE                                               6
21/tcp  open  ftp                                                   7
22/tcp  open  ssh                                                   8
23/tcp  open  telnet                                                9
80/tcp  open  http                                                  10
443/tcp open  https                                                 11
MAC Address:08:00:27:33:11:8D (Oracle VirtualBox virtual            12
   NIC)
Nmap done: 1 IP address (1 host up) scanned in 0.69                 13
   seconds
```

Lastly, telnet port was opened and initiated from the Debian (Kali) side, this connection attempt was acquired. However, no valid data were extracted, except for the originating IP address and time, moreover the following log message was an unrecognizable string of characters. Nevertheless, this intrusion could be defined as captured.

To summarize the PenTBox Suite, it served as a stepping stone to a better understanding of what log messages can be captured by a Honeypot. It was not however a viable tool for the purpose of this thesis, for it's limiting ability of configuration, the fact that multiple terminals needed to be running at the same time to cover multiple service ports and the overall simplicity of service emulation. Moreover, PenTBox does not log the attacker's commands in the Ubuntu shell.

3.2 Deployment of Cowrie

As previously described in the Cowrie section, Cowrie is a medium to high interaction SSH and Telnet Honeypot, with he ability to function in two states. Either as an emulated shell, which is it's default state, where it imitates a Debian 5.0 installation filesystem or as SSH and Telnet proxy.

Installation was done on a clear image of 64-bit Linux (Ubuntu) with 1 CPU core, dedicated 4 GB of RAM and 15 GB of storage.

Cowrie can be deployed by several methods, for the purpose of this thesis two were chosen, installation with virtual environment and with the Docker engine. Next part will describe the virtual environment method of installation.

First comes the installation of required dependencies, this can be done for both python3 and python2 based environment:

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install git python
   -virtualenv libssl-dev libffi-dev build-essential
   libpython3-dev python3-minimal authbind virtualenv
```

It is strongly recommended to run the Cowrie Honeypot with a dedicated non-root user ID:

```
ubuntu@ubuntu-VirtualBox:~$ sudo adduser --disabled-
   password cowrie
Adding user 'cowrie' ...
[...]
ubuntu@ubuntu-VirtualBox:~$ sudo su - cowrie
```

Download Cowrie clone from the master repository:

```
git clone http://github.com/cowrie/cowrie
Cloning into 'cowrie'...
[...]
Checking connectivity... done.
ubuntu@ubuntu-VirtualBox:~$ cd cowrie
```

Virtual environment set-up:

```
virtualenv --python=python3 cowrie-env
New python executable in ./cowrie/cowrie-env/bin/python
Installing setuptools, pip, wheel...done.
```

Virtual environment activation and package installation with `pip`:

```
source cowrie-env/bin/activate
(cowrie-env) $ pip install --upgrade pip
(cowrie-env) $ pip install --upgrade -r requirements.txt
```

Cowrie initialization:

```
bin/cowrie start
Activating virtualenv "cowrie-env"
Starting cowrie with extra arguments [] ...
```

When installed by the Docker engine, the set-up is as followed.

Cowrie image pull and subsequent initialization via the Docker engine:

```
ubuntu@ubuntu-VirtualBox:~$ docker pull cowrie/cowrie
[...]
ubuntu@ubuntu-VirtualBox:~$ docker run -p 2222:2222 cowrie/
   cowrie
```

After the initialization, Cowrie Honeypot starts with the following message:

```
[...]
[-] Generating new RSA keypair...
[-] Generating new DSA keypair...
[-] Ready to accept SSH connections
[...]
```

To get a grasp of the initialization message, DSA and RSA standards will be briefly explained. DSA (Digital Signature Algorithm) is a key pair generation mechanism and it's security is based on the discrete logarithm problem. It is faster for signature generation but on the other hand slower for validation, slower when encrypting but faster when decrypting and security can be considered equivalent compared to an RSA key of equal key length. RSA (Rivest–Shamir–Adleman) can be considered to be one of the first key cryptosystem generator. The security of the RSA algorithm is based on the factorization of large integers. This however exceeds the purpose and the assignment of the thesis and will be therefore abandoned.

Nevertheless, after the generated key pairs, we probe the Cowrie from the Kali side, to see the capabilities of the Honeypot. This process and the data provided by the system are visualized in the following subsection Benchmarking of Cowrie, where it will be described. The scan was done in few steps, starting with `ping` to see if the host is reachable, then continuing with the `nmap` tool to see if the Cowrie

Honeypot has enabled the desired ports. Cowrie Honeypot was then engaged from Kali side with `ssh 10.0.2.15` command. After the SSH connection was established, we tested what directories can be found in the Cowrie Honeypot with the `ls -all` command shown in the Listing 3.11 below.

Listing 3.11: Cowrie directories of the emulated Debian 5.

```
root@svr04:~# ls -all                                                    1
drwx------ 1 root root 4096 2013-04-05 12:25 .                           2
drwx------ 1 root root 4096 2013-04-05 12:25 ..                          3
drwx------ 1 root root 4096 2013-04-05 11:58 .aptitude                   4
-rw-r--r-- 1 root root  570 2013-04-05 11:52 .bashrc                     5
-rw-r--r-- 1 root root  140 2013-04-05 11:52 .profile                    6
drwx------ 1 root root 4096 2013-04-05 12:05 .ssh                        7
```

We then `cat /etc/passw` to show the text file containing the attributes of users and accounts running on the Linux machine, in this case the emulated Honeypot system. To finish up the test of the Cowrie session we then tested `wget` and received a response according to the predetermined Cowrie behavior which was described in the Cowrie section.

The session was terminated automatically after 200 seconds. This process was captured with the `nmon` utility which is a performance system monitor tool for the Linux and AIX operating systems and will be covered in the following subsection.

3.2.1 Benchmarking of Cowrie

As mentioned previously, the benchmark test of the Cowrie Honeypot session was captured with the utility `nmon` and subsequently visualized with the help of `nmonchart` utility extension.

We initiated the `nmon` capture before the initialization of the Honeypot system to gather the whole session from the start to finish. Starting with the CPU Utilization Percentage visualization.

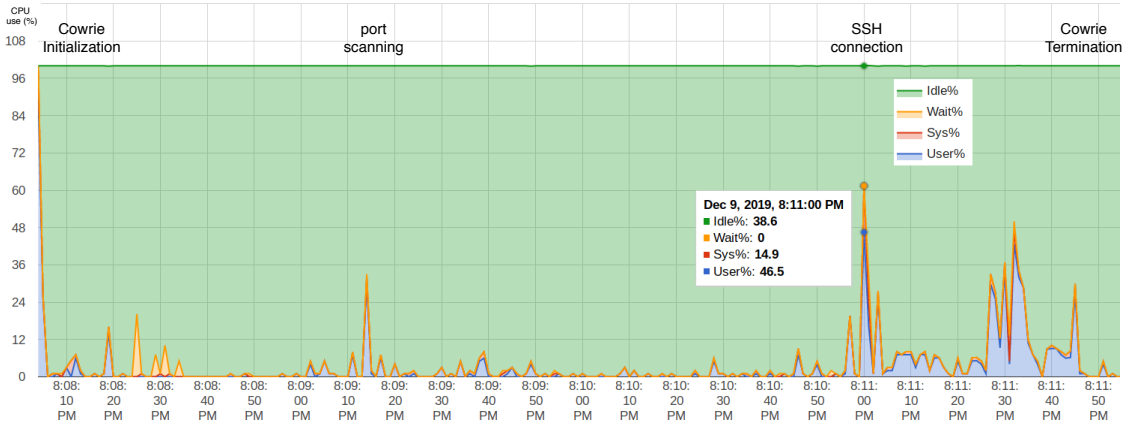

Fig. 3.1: Crowrie, CPU Utilization Percentage.

From the course of the CPU Utilization Percentage visualization we can see that after the initialization of the Cowrie Honeypot the use drops after it peaked while generating the RSA and DSA key pairs. Small CPU usage can be seen shortly after the initialization and `[-] Ready to accept SSH connections` message, which is attributed to the `ping`, followed by the `nmap` tool. Ultimately, we move onto the connection with the Honeypot itself which is also highlighted in the Fig. 3.1, where the Honeypot and the client use the Diffie-Hellman Key Exchange Algorithm to create a symmetrical key, followed by access request and authentication and ultimately the Honeypot engages in the SSH connection. After the connection establishment, the Honeypot has to respond to the Kali side and to also store and show the shell commands. Lastly the final peak of the visualization is the Cowrie session termination.

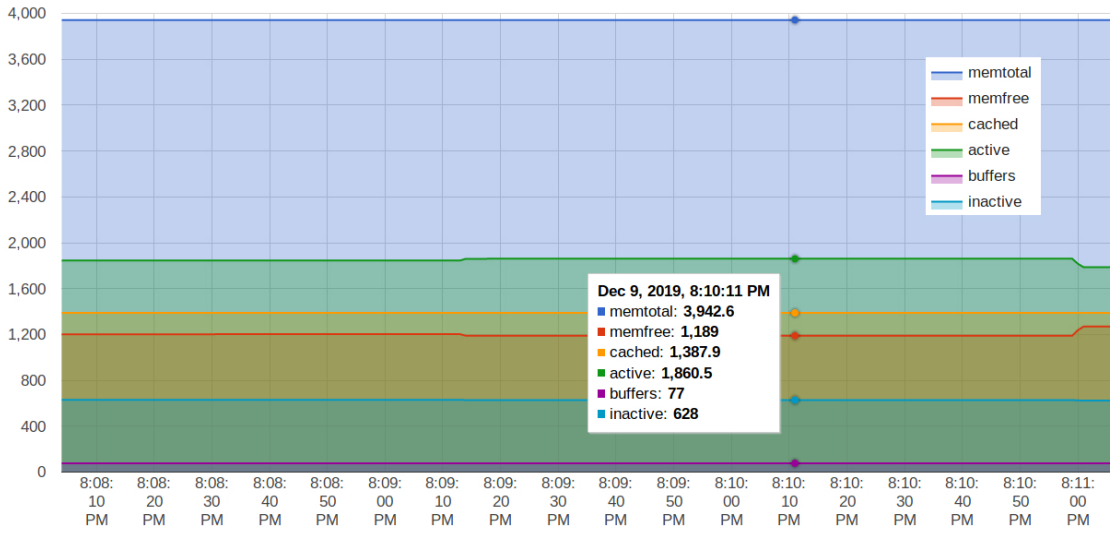

Fig. 3.2: Cowrie, Real Memory, RAM in MB.

Only visible difference in the Real Memory, RAM in MB visualization is the moment the `nmap` tool was initiated, no increased can be spotted even when the SSH connection is established. However, we can see a slight decrease of active memory after the Honeypot termination.

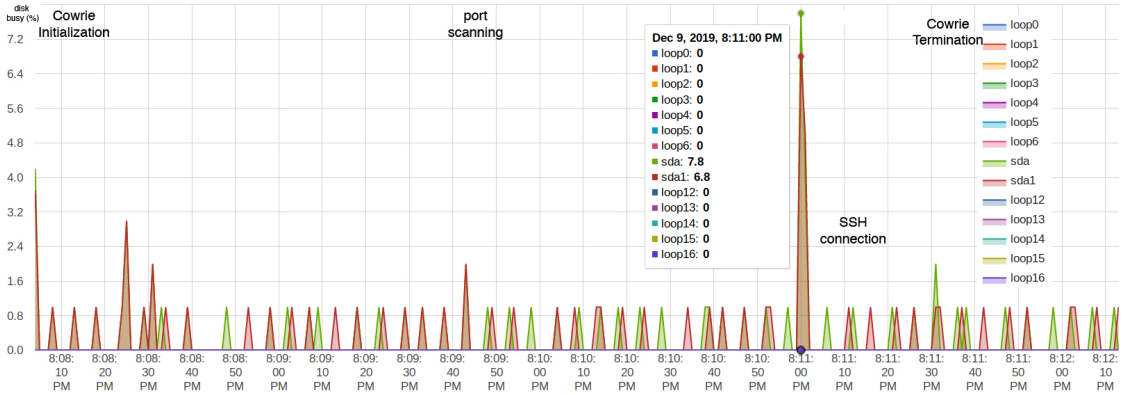

Fig. 3.3: Cowrie, Top 15 disks by sum (Busy percentage).

The Disk Busy percentage monitors the percentage of elapsed time when the disk was busy processing a read or write request. The single significant event is visualized during the SSH connection establishment. We can notice `loop` entries, these are pseudo file systems that allow mounting of a file to act as if it was a partition. The available space on these `loop` entries will always stay at the value 0, because they are not classified as a file system.

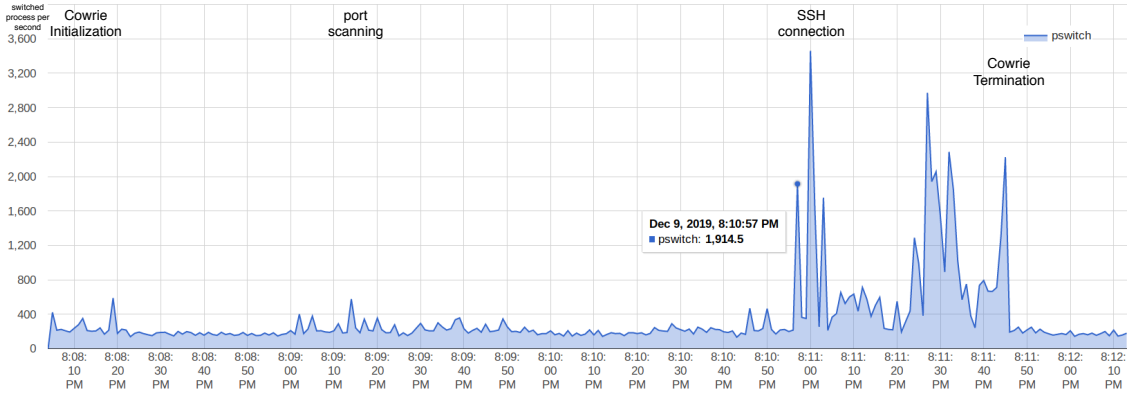

Fig. 3.4: Cowrie, Process switches per second.

Finally, moving onto the Process switches per second visualization. The `pswitch` value indicates the number of context switches rates per second. A context switch, sometimes referred to as a task switch or a process switch, is the switching of the CPU from one process or thread to another. This visualization resembles the CPU Utilization Percentage visualization rather closely, this may be caused by the fact

that the virtual machine has a single core dedicated to it, if there were more cores however, this statistic could potentially change.

To summarize the Cowrie Honeypot benchmarks, their accuracy may vary in comparison to a non emulated system, moreover we can assume the precision of the values would change if run on a non emulated system. Furthermore, network tests are not included in this part, for the interaction with the Cowrie Honeypot was done from one source (Kali). Furthermore, there were no bandwidth stress test carried out. There is however, motivation to further test the environment with attacks such as brute force attack or Man in the Middle attack and further evaluate the system and to deploy the Cowrie Honeypot for a prolonged period of time and analyze the incoming traffic.

3.3 Deployment of Conpot

As previously mentioned in Conpot section, Conpot is a low-interaction Industrial Control System Honeypot that covers a range of commonly used industrial control protocols and is able to emulate complex system infrastructures.

Installation was done on a clear image of 64-bit Linux (Ubuntu) with dedicated 4 GB of RAM and 15 GB of storage.

Conpot emulates, by default, a rough simulation of basic Siemens SIMATIC S7-200 Programmable Logic Controller (PLC) device CPU with 2 slaves.

First installation attempt failed during the Conpot setup, most likely because of the interference of dependencies from the Honeyd installation. For that reason it was determined that clear image would be ideal to deploy each Honeypot as mentioned at the beginning of this chapter.

After further research the best way of installation was determined to be with the help of a Docker Engine.

`apt` package index udpate:

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get update
```

1

Package install to allow the `apt` to use a repository over HTTPS:

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install apt-
   transport-https ca-certificates curl gnupg-agent
   software-properties-common
```

1

Add Docker's GPG key:

```
ubuntu@ubuntu-VirtualBox:~$ curl -fsSL https://download.
   docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Verify the presence of the key:

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-key fingerprint 0
   EBFCD88
```

Stable repository set-up:

```
ubuntu@ubuntu-VirtualBox:~$ sudo add-apt-repository
"deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
```

Followed by installation of Docker's latest version:

```
ubuntu@ubuntu-VirtualBox:~$ sudo apt-get install docker-ce
```

After the installation of the Docker Engine a test command was initiated `sudo docker run hello-world`, which informed about the proper functionality of the Docker Engine. The decision was made to build the Conpot Honeypot from a Docker source image, for easier manipulation of the configuration file and overall inspection of the Honeypot partitions. The build of the image from source was following.

Clone the repository from source:

```
ubuntu@ubuntu-VirtualBox:~$ git clone https://github.com/
   mushorg/conpot.git
```

Enter the cloned directory:

```
ubuntu@ubuntu-VirtualBox:~$ cd conpot/docker
```

Build the Conpot image, with the default template, named `conpot:default` with following command:

```
ubuntu@ubuntu-VirtualBox:~$ docker build -t conpot:default
   .
```

3.3.1 Structure of Conpot

This subsection lists the main individual components of the master branch of Conpot release 0.6.0. The root directory includes the texts files bearing information about the Conpot Honeypot independently in files, `Changelog.txt`, `LICENSE.txt`,

`Makefile` for docker initialization, `requirements.txt`, `README.rst` and the required installation scripts. Furthermore, the root directory contains subdirectories: `bin, conpot, docker, docs`.

The `bin/` directory includes Conpot's main running script. Apart from that, this subdirectory contains Conpot additions, expanding the Conpot's functionality such as `conpot-cloner` which fixes the problem with Conpot Management Information Base (MIB) cloner's methods and others [37].

The `conpot/` directory includes the primary configuration file and a number of subdirectories holding the implementations of emulators, source codes for protocol's functionality, templates for the emulation of various devices, testing suites and utilities.

The `docker/` directory stores `Dockerfile` for the image assembly and `docker-compose.yml` a human-readable data-serialization language YAML file that defines networks, services and volumes. Service definition bears applied configuration to each container started for related service, similar to passing a command line commands to the docker container create. The `docs/` directory includes the documentation files and all the necessities for creating and maintaining Conpot's documentation

Conpot was triggered by following command, where as an example `-p 80:8800`, `80` is a localhost port and `8800` is a Docker port. By this command a Docker port is redirected to a localhost port and is now available.
Next we initiate `-network=bridge` to bridge our network. Finally we initialize a local image of the Conpot with a default template by `conpot:default`.

Listing 3.12: Conpot Initialization.

```
ubuntu@ubuntu-VirtualBox:~/conpot/docker$ sudo docker run -
   it -p 80:8800 -p 102:10201 -p 502:5020 -p 161:16100/udp
   -p 47808:47808/udp -p 623:6230/udp -p 21:2121 -p
   69:6969/udp -p 44818:44818 --network=bridge conpot:
   default
```

For the purpose of saving space there will not be presentation of the whole start-up sequence. However, lines of interest or lines bearing a greater significance will be shown and analyzed in more detail. Conpot initialization output informs us about several settings by which it was triggered and set-up. Path to the source file that is being triggered is `/conpot/bin/conpot`.

Listing 3.13: Conpot Template Initialization.

```
2019-11-23 21:28:24,352 Starting Conpot using template: /
   home/conpot/.local/lib/python3.6/site-packages/conpot
   -0.6.0-py3.6.egg/conpot/templates/default
```

Conpot that informs about the paths to various files, which are executed upon it's initialization, such as the configuration file, protocol templates or virtual file system creation. A curious part was the fetched external IP address: `91.127.212.114`, which redirected to a ZTE Speedport Entry 2i router.

Following onto the enabled protocols:

Listing 3.14: Conpot Protocols Initialization.

```
2019-11-23 21:28:24,704 Found and enabled modbus protocol.
,714 Found and enabled s7comm protocol.
,718 Found and enabled http protocol.
,720 Found and enabled snmp protocol.
,767 Found and enabled enip protocol.
,743 Found and enabled ipmi protocol.
,922 Found and enabled ftp protocol.
,004 Found and enabled tftp protocol.
```

It would appear that Conpot Honeypot enables all protocols that are supported, despite simulating a device that excludes some of the enabled protocols. This assumption was established after the testing of a different template that simulated a Guardian AST tank-monitoring system. Furthermore, protocols requiring data storage (FTP, TFTP), create persistent data stores. Another of the presented messages is the proxy state, either configured and enabled or unconfigured and stopped.

Listing 3.15: Conpot Port Initialization.

```
2019-11-23 21:28:25,010 Modbus server started on:
   ('0.0.0.0', 5020)
,016 S7Comm server started on: ('0.0.0.0', 10201)
,018 HTTP server started on: ('0.0.0.0', 8800)
,188 SNMP server started on: ('0.0.0.0', 16100)
,193 Bacnet server started on: ('0.0.0.0', 47808)
,194 IPMI server started on: ('0.0.0.0', 6230)
,197 FTP server started on: ('0.0.0.0', 2121)
,198 Starting TFTP server at ('0.0.0.0', 6969)
```

Lastly, indications of started services. Considering the context of servers, 0.0.0.0 means all IPv4 addresses on the local machine. The IP address of the virtualized Ubuntu machine is `10.0.2.15` and we should be able to test it's accessibility by typing `localhost`, `0.0.0.0` or `10.0.2.15` to the internet browser. This will also serve as a first look at what logs can Conpot produce.

Listing 3.16: HTTP session log.

```
2019-11-23 21:29:07,620 New http session from 172.17.0.1 (   1
   f78bb4eb-8602-4219-b22d-0616478da1b6)
,621 HTTP/1.1 GET request from ('172.17.0.1', 57710): ('/',   2
    [('Host', 'localhost'), ('User-Agent', 'Mozilla/5.0 (
   X11; Ubuntu; Linux x86_64; rv:68.0) Gecko/20100101
   Firefox/68.0'), ('Accept', 'text/html,application/xhtml+
   xml,application/xml;q=0.9,*/*;q=0.8'), ('Accept-Language
   ', 'en-US,en;q=0.5'), ('Accept-Encoding', 'gzip, deflate
   '), ('Connection', 'keep-alive'), ('Upgrade-Insecure-
   Requests', '1')], None). f78bb4eb-8602-4219-b22d-0616478
   da1b6
,622 HTTP/1.1 response to ('172.17.0.1', 57710): 302.        3
   f78bb4eb-8602-4219-b22d-0616478da1b6
```

Line 1 indicates that a new HTTP session was initiated from source, in this case the virtual machine itself, followed by what appears to be some form of session fingerprint, for it's structure does not alter during the capture for this specific service session. The following lines are identical to the PenTBox HTTP session and do not require further analysis, for full description can be found under the List 3.3. The structure differentiates only in the JSON (JavaScript Object Notation) formatting. Lastly, line number 3 initiates a HTTP response number 302, which informs that the requested address was found. Moreover, `302 Found` indicates that the requested resource was moved to a temporary URL given by the `Location` header. The internet browser redirects to this page does not update the links to the resource.

Moving onto the FTP connection capabilities, we initiate the ftp connection from the Kali virtual machine to our Conpot Honeypot as shown in the List 3.17.

Listing 3.17: FTP session initialization, Kali side.

```
ubuntu@ubuntu-VirtualBox:~$ ftp                              1
ftp> open 10.0.2.15                                          2
Connected to 10.0.2.15                                       3
200 FTP server ready.                                        4
```

After a successful connection is established we receive `200 Series` response, meaning the requested action has been successfully completed. This message is generated from the Conpot Honeypot to the requester.

`\r` (Carriage Return) - moves cursor to the beginning of the line without advancing to the next line.

`\n` (Line Feed) - moves cursor down to next line without returning to the beginning of the line, in the Unix environment `\n` moves to the start of the line.

\r\n (End Of Line) - a combination of \r and \n.

Listing 3.18: FTP session initialization, Conpot side.

```
2019-11-26 18:10:04,247 New ftp session from 10.0.2.4 (9
   c4273d2-d5c6-417e-9a28-e28e3ff11d53)
,249 New FTP connection from 10.0.2.4:37510. (9c4273d2-d5c6
   -417e-9a28-e28e3ff11d53)
,250 FTP traffic to ('10.0.2.4', 37510): {'response': b'200
    FTP server ready.\r\n'} (9c4273d2-d5c6-417e-9a28-
   e28e3ff11d53)
```

To better understand the Conpot FTP server capabilities, following exchange took place from the Kali virtual machine. To justify the short length of the FTP session, the reason for that was that the session was automatically closed after a while of detected inactivity. Furthermore the time was session was stretched to the best of our ability.

Listing 3.19: FTP session initialization.

```
Name (10.0.2.15:ubuntu): root
331 Now specify the Password.
Password:
530 Authentication Failed.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /home/ubuntu/
530 Log in with USER and PASS first.
ftp> user user
331 Now specify the Password.
Password:
530 Authentication Failed.
Login failed.
ftp> exit
221 Bye.
```

After the successful connection establishment to the Conpot's FT server we are requested to insert user name. To this request we are presented with `300 Series` response, meaning that the command has been accepted, but the requested action is on hold, pending further information. Followed by the password request insertion, we are presented with the `500 Series` response, indicating a syntax error. signalizing that either the command was unrecognized and the requested action did not take

place or that the command line was too long. Nevertheless, the connection was not terminated, continuing with the FTP test we attempt show available directories with `ls -all`, this however fails and requests access with user name and password. If however, we attempt to change directory by `cd` followed by the tab to suggest a path, directories of the Ubuntu system are shown. This clearly shows that the connection was established and the responses are not just emulated. If we terminate this connection we are presented with `221` response meaning the service is closing the control connection.

Moving onto the terminal printout with the interest of higher significance, the Conpot Honeypot side, to examine what actions were logged. Recurring instances will be replaced with `[...]`, for reasons of space saving and the immaterial information value of such instances. The let-go parts of the printout are shown only in the first two lines of the printout: client's IP address `('10.0.2.4', 37510)` and the fingerprint `(9c4273d2-d5c6-417e-9a28-e28e3ff11d53)`.

Listing 3.20: FTP session log.

```
,193 FTP traffic to ('10.0.2.4', 37510): {'request': b'
   USER user\r\n'} (9c4273d2-d5c6-417e-9a28-e28e3ff11d53)
,195 Received command USER : USER user from FTP client
   ('10.0.2.4', 37510): [...]
,196 FTP traffic to [...] {'response': b'331 Now specify
   the Password.\r\n'} [...]
,328 FTP traffic to [...] {'request': b'PASS user\r\n'}
   [...]
,334 Received command PASS : PASS user from FTP client
,335 FTP traffic to [...] {'response': b'530
   Authentication Failed.\r\n'} [...]
,344 FTP traffic to [...] {'request': b'SYST\r\n'} [...]
,346 Received command SYST : SYST from FTP client [...]
,348 FTP traffic to [...] {'response': b'215 UNIX Type: L8
   \r\n'} [...]
2019-11-26 18:10:21,938 FTP traffic to [...] {'request': b
   'CWD /home/ubuntu/\r\n'} [...]
,940 Received command CWD : CWD /home/ubuntu/ from FTP
   client
,204 FTP traffic to [...] {'request': b'QUIT\r\n'} [...]
,205 Received command QUIT : QUIT from FTP client [...]
,207 FTP traffic to [...] {'response': b'221 Bye.\r\n'}
```

Listing 3.21: FTP session Conpot log.

```
FTP statistics for client     : ('10.0.2.4', 47122)          1
-----------------------------------                          2
Logged in as user :root with uid: None                       3
Failed login attempts         : 2/3                          4
Start time                    : Mon Dec  9 19:55:46 2019     5
Last active on                : Mon Dec  9 19:58:10 2019     6
-----------------------------------                          7
Total data transferred      : 244  (bytes)                   8
Command channel sent        : 177  (bytes)                   9
Command channel received    : 67  (bytes)                    10
Data channel sent           : 0  (bytes)                     11
Data channel received       : 0  (bytes)                     12
```

Based on the Listing 3.20 above, we can analyze similar message output spanning over all lines. The captured shell commands did not however match with the commands from the Kali side. Furthermore, line 7 indicates that there was a `SYST` request asking for information about the server's operating system. Thus, server accepted this request with code `215 UNIX Type: L8`. Following onto the line number 11, we can see that the change of directory `CWD` attempt was captured successfully. Also the insertions of username and password was captured, despite not providing the exact input but rather a FTP command. As shown in the Listing 3.22, Conpot is able to generate a printout of the FTP session, this log printout was however not captured over all testing sessions. The testing was carried out in short time spans, for the sessions were being terminated after a brief moment of inactivity. No actual data, except the command channel sent/received were transferred, for the previously mentioned issue with access credential insertion. This behavior could potentially give away, if encountered by experienced attacker, that it is not a real system or that the functionality of the FTP server is somewhat unreliable.

Following onto the `nmap` test printouts. The `nmap`, used to discover hosts and services on a network by sending packets and analyzing the responses, would not be as interesting on it's own. However, during the `nmap` session targeting the Conpot Honeypot, the process seemed to run into resolving issues. This event will be described and further evaluated below.

Listing 3.22: Nmap Conpot.

```
2 services unrecognized despite returning data.                    1
=====NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)=====           2
SF-Port80-TCP:V=7.60%I=7%D=11/26%Time=5DDD5797%P=x86_64-pc         3
   -linux-gnu%r(Get
SF:Request,7E,"HTTP/1\.1\x20302\x20Found\r\nDate:\x20Tue,\         4
   x2026\x20Nov\x20
SF:2019\x2016:40:11\x20GMT\r\nContent-Type:\x20text/html\r         5
   \nLocation:\x20/
[...]                                                              6
SF:\x20found\x20-\x20Siemens,\x20SIMATIC,\x20S7-200</TITLE         7
   >\n\x20\x20\x20\
SF:x20</HEAD>\n\n\x20\x20\x20\x20<BODY>\n\x20\x20\x20\x20\         8
   x20\x20\x20\x20<
SF:h2>CP\x20443-1\x20EX40</h2>\n\x20\x20\x20\x20\x20\x20\         9
   x20\x20<hr>\n\x20
SF:0This\x20resource\x20could\x20not\x20be\x20found\.<br>\         10
   n\x20\x20\x20\x2
SF:0</BODY>\n\n</HTML>");                                          11
```

The `nmap` did not work on every occasion, returning a `Segmentation fault` error. However, during the successful scan of the network it did return the presence of the emulated Siemens SIMATIC S7-200. We can notice that at the beginning of the `nmap` readout there are `2 services unrecognized despite returning data`. We would assume that the reason for that is that the `nmap` capabilities do not stretch to some of the emulated ICS protocols. Furthermore the `nmap` outputs on the Conpot side and fingerprints generated from the Honeypot, on the Kali side, were fluctuating in the information value. The reason for that might be in the inability of the Conpot to respond to certain requests, requiring further evaluation. In conclusion, we were able to identify what device was being emulated, despite the uncertainty of the exact number and specification of opened ports, in respect to the `nmap` capability set. Detected ports were 21, 80, 631.

3.3.2 Benchmarking of Conpot

As mentioned previously, the benchmark test of the Conpot Honeypot session was captured with the utility `nmon` and subsequently visualized with the help of `nmonchart` utility extension. Both of which were covered earlier and will not be further described. Starting with the CPU Utilization Percentage visualization graph show in the following figure.

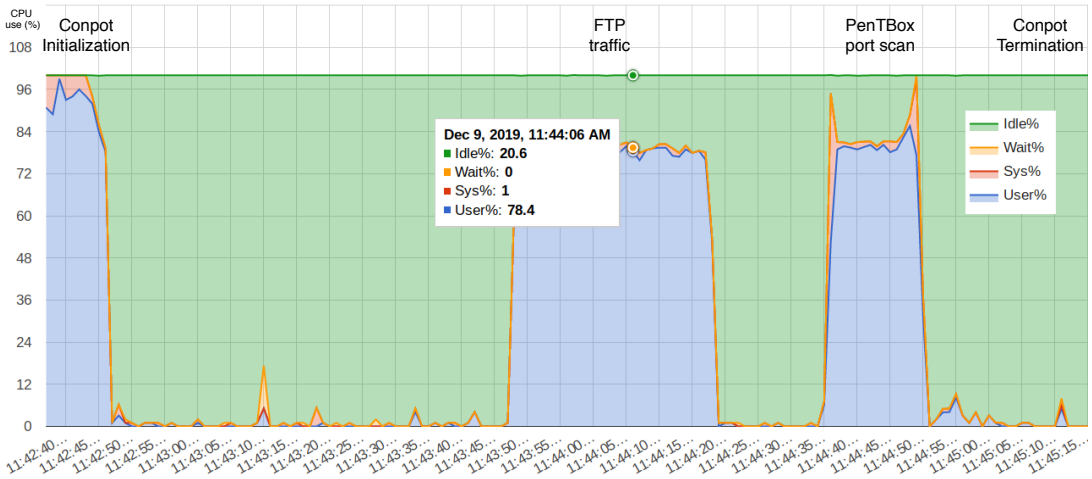

Fig. 3.5: Conpot, CPU Utilization Percentage.

In comparison to the CPU Utilization Percentage visualization of the Cowrie Honeypot, the Conpot Honeypot requires higher CPU utilization for a longer period of time, this is cause by the initialization of servers, which is shown in the Fig. 3.16. After the initialization is completed, we `ping` the Conpot system, followed by the `nmap` tool to see what ports are open. As previously mentioned we ran into resolving issues during the `nmap` port scan, which did not occur during this test session. Moving onto the middle part of the visualization, which is also highlighted, we initiate the FTP server and carried out sequence which was shown in the Fig. 3.19. From what we can evaluate the FTP session does put a significant strain on the system, considering the resources available to the system itself. Finally, we initiated port scan from the PenTBox utility.

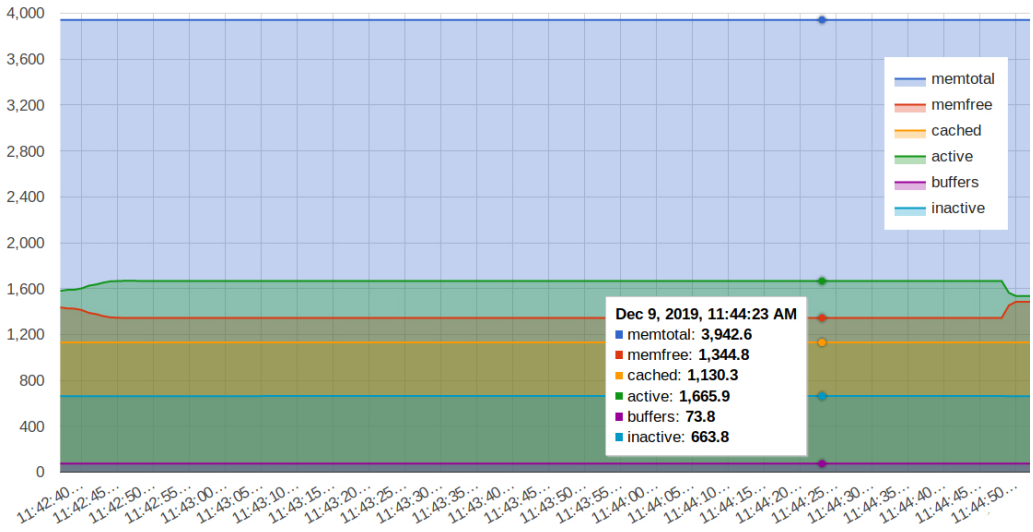

Fig. 3.6: Conpot, Real Memory, RAM in MB.

Moving onto the Real Memory, RAM in MB visualization. We can clearly notice the memory allocation after the Conpot initialization, in contrast to the Cowrie, where the memory was allocated after a certain period of the Honeypot initialization. Moreover, the memory allocation drop is visible after the Conpot termination as well.

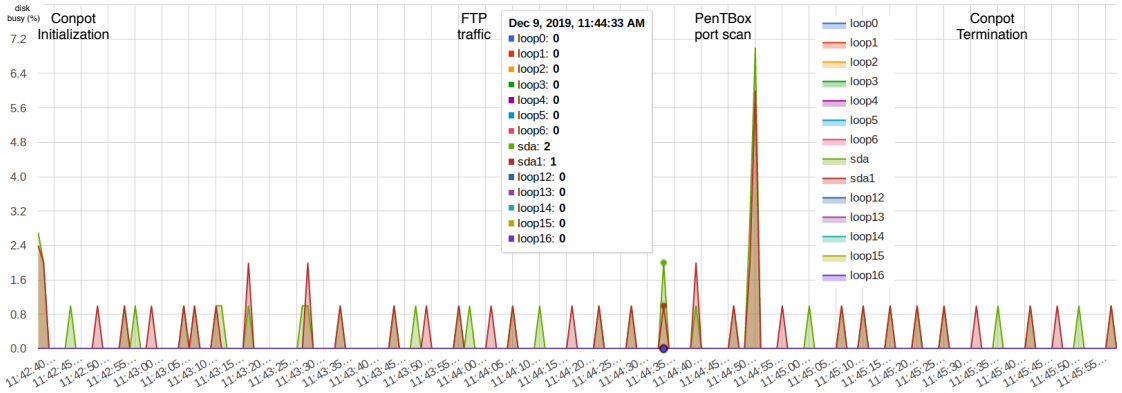

Fig. 3.7: Conpot, Top 15 disks by sum (Busy percentage).

The Busy percentage visualization graph shows increased values only at the beginning of the PenTBox port scan and a peak after the end of the port scan. Slight increase is visible during the session start itself. Other events, such as Conpot initialization and `nmap` are slightly visible, except the FTP connection and Conpot termination that did not produce values of significance.

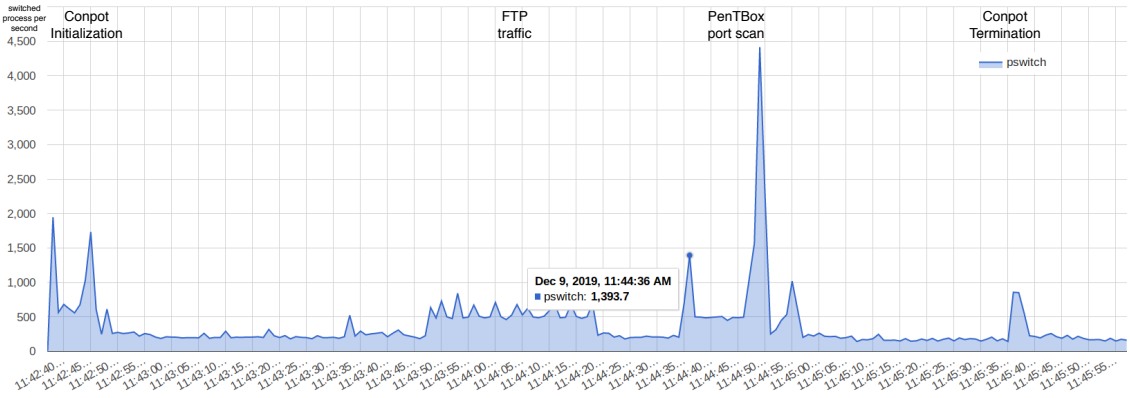

Fig. 3.8: Conpot, Process switches per second.

In the process switches per second visualization we can clearly determine the whole session from start to finish. Based on the previous process switches visualization, we would assume a significantly higher increase during the FTP session. However, this may be cause by the simple nature of the interaction, where we were unable to actually transfer any data between the systems. We have produced two other graph visualizations, `Run Queue: running, ready or blocked processes`

and `Fork() and Exec() system calls per second`, which did not however provided any values of significance and were therefore scratched of the test event list. As previously mentioned during the Cowrie test session, network traffic visualization is not included for there was only one host interacting with the Honeypot.

As stated earlier, these visualization assumingly do not produce most precise data, this fact could be changed by implementing the Honeypot system onto a non virtualized system.

Furthermore, the Conpot Honeypot is not able to detect all traffic, for this reason a additional software should be used as a part of the whole system such as `Snort`, `Suricata` or `Wireshark`. Only two of these tools were used, `Wireshark` and `Snort`, for isolated test events and their data was only analyzed briefly, to get a better understanding of what intrusion traffic they are able to detect.

3.3.3 Template extension and protocol support extension

If one would desire to extend the functionality of the Conpot Honeypot either by additional protocol implementation or by full-on device emulation, it can be done by implementing the respective protocol functionality into `conpot.protocols` package or by developing and extending templates that are being triggered upon the start of the Conpot Honeypot.

As the Conpot documentation suggests, `conpot.protocols` package includes several subpackages with implemented protocols divided into modules and submodules. The Conpot Honeypot is flexible enough for it to be modified to simulate other models of ICS SCADA devices. The majority of the configuration options are stored in `default.xml` file, found in the templates/directory off of the directory where Conpot was installed.

The first step would be modifying the `default.xml` file. In order to understand and to know what modification are required we should know how the emulated device presents itself on the network system. This could be done by researching an actual device which we want to emulate, `nmap` tool could serve the purpose of discovering the attack surface.

Monitoring of the traffic and optional alerting of the user from the Honeypot system can be achieved. Conpot provides the option of log sending to a remote Syslog server. One way of viewing and scrubbing through the logs could be the use of a Splunk Enterprise tool, which also alerts the user of the hostile activity and indexes the logs. This would allow the Honeypot system operators to quickly and easily search through the Honeypot activity, thus enabling the security operators to respond during intrusion events. To summarize, further analysis needs to be carried out to successfully develop and deploy the desired system.

3.3.4 Evaluation of available Honeypot solutions

Based on the tests of the Honeypot solutions in previous chapters, a conclusion was made that each of the aforementioned Honeypot solution does not meet the requirements of a High-interaction Honeypot. Thus a development of a new Honeypot is required. The following Tab. 3.1 shows a brief summary of functionality, respected strengths and weaknesses of analyzed Honeypots.

Name	Level of Interaction	Main Function	Strengths	Weaknesses
PentBox	Low	Security tool with Honeypot feature as request/response	- Ease of use	- Not a pure Honeypot - No ICS application
Honeyd	Low	Multiple emulated hosts simulating network services	- Ease of use - Proxy capabilities	- Scalability - Abandoned project - No ICS application
Conpot	Low	Honeypot emulating ICS services	- Simple configuration - Scalability - Honeynet capability - Support of multiple ICS protocols	- Log capabilities - Limited service emulation - Low interaction nature
Cowrie	Medium/High	Emulates SSH and telnet services	- Ease of use - Emulated shell	- No service emulation
HonSSH	High	Acts as a SSH proxy between the attacker and high-interaction Honeypot	- Simple configuration - Password spoofing - Separates SSH connection	- Requires a Honeypot to be attached

Tab. 3.1: Brief Honeypot solutions analysis.

Based on the traits of previously mentioned Honeypot solutions, the decision was made to develop a new High-interaction Honeypot, that would be able to simulate real services, ICS communication protocols with the ability to facilitate a network which could be expanded based on the developers needs. Furthermore, the SSH Honeypot solution HonSSH is to be included in this Honeypot development, for it´s ability to act as a SSH proxy between the attacker and a High-interaction Honeypot, this will be described in further detail in following chapter.

4 Development of a new High-interaction Honeypot

This chapter will focus on the development and deployment part of new High-interaction Honeypot, it´s structure, capabilities, general purpose, side of emulation, underlying hardware and the deployment of the whole system onto a clean VPS (Virtual Private Server) service.

Moreover, the technology applied and partitions forming the Honeypot solution will be described in the following sections. The technology used in this Honeypot solution was described in the analytic part of this thesis. Thus, this chapter will no describe the technology used, but rather elucidate the implementation of the technology used to meet the needs of a fully functioning High-interaction honeypot.

4.1 High-interaction Honeypot structure

As previously mentioned in the analytic part of this thesis, based on previous testing and evaluation of some of the Honeypot solutions, the decision was to build on the security research toolkit for Industrial Control System security, MiniCPS.

This platform will emulate protocol services running on the emulated MiniNet topology, connecting it to the main interface of the Virtual Private Server with static routing through a virtual switch which will be emulated by MiniNet as well. The following Fig. 4.1 illustrates a basic structure of the new High-interaction Honeypot.

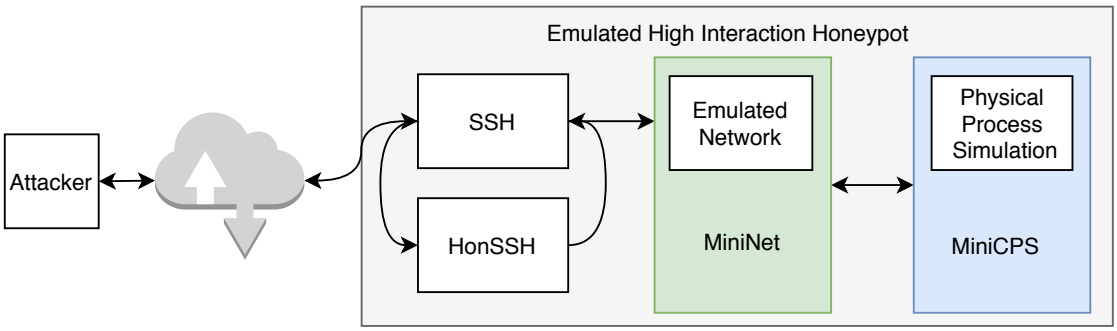

Fig. 4.1: Concept High-interaction Honeypot structure

To access this emulated network, SSH port 22 will be exposed to the internet. Moreover, HonSSH listens to all incoming traffic to this port. Through port forwarding, the attacker if forwarded to the Docker container that exposes port 2222 to the main VPS interface, more on this in the following section Docker Containerization. Moreover, Docker containerization was not the first option of restricting the

attacker to a designated environment, initially the `chroot` feature was supposed to serve this purpose, due to complications this solution was abandoned.

One of the goals of this thesis is the implementation of two Industrial Control Protocols into the High-interaction Honeypot, Modbus and Ethernet/IP (ENIP) protocols, by emulating communication. During the development and implementation of these protocols, no real PLC could not have been analyzed, however emulation of sensors attached to simulated processes was achieved nevertheless and subsequent interaction with the master of the respected protocols.

This thesis provides scenarios, one for each of the aforementioned protocols. The interfaces on which they are the processes running and the simulated topology is visible to the attacker and the processes simulated or rather the packets generated by these processes can be captured by the intruder, moreover the attackers can plant their own packets, on which the master node in the topology will acknowledge these requests.

`PyModbus` library was utilized to simulate the Modbus protocol. This library is written in Python and provides a full Modbus protocol implementation by utilizing twisted for its asynchronous communications core. No additional dependencies are required for this library to function [38].

`CPPPO` implements a subset of the EtherNet/IP protocol utilized by some industrial control equipment. It is also written in the Python language [39].

The MiniCPS includes the aforementioned SWaT (water treatment) example in the master directory. For this thesis temperature sensor emulation was developed for both scenarios, based on `random.uniform(a,b)` which returns random floating number N for which applies that `a<=N<=b` for `a<=b` and `b<=N<=a` for `b<a` [40]. This translated into the temperature value to fluctuate, after configuring the interval, in range of a 1 degree around the final value.

The MiniCPS framework could be expanded further to supplement other Industrial Control System protocols by utilizing libraries supporting these protocols such as SCAPY for Profinet protocol, OpenDNP3 for the DNP3 protocol, etc. Moreover, protocol functionality depends on the quality of the implemented open-source library. Any additional library extension is done in the module
`minicps/protocols.py`, where the base class is the protocol and additional newly implemented `[NewProtocolName]Protocol(Protocol)` inherits from the Protocol class as a child class.

4.1.1 Docker Containerization

Even though, this thesis work does not utilize Docker to facilitate the structure, Docker containerization plays a significant role in isolating the environment that

the attacker can access. Initially, this was supposed to be achieved by utilizing feature called `chroot` or `chroot jail`. This feature forms a separate, apparent root directory environment to which the attacker is bound and can not access the outside world of the Linux filesystem and commands, thus restricting, jailing the attacker to prevent the harm possibility to minimum. Initially, all SSH connections were to be chroot-jailed by utilizing the OpenSSH´s `ChrootDirectory` feature. Upon further inspection this aim was abandoned due to the limited author´s understanding of the process behind this feature.

Furthermore a more liable solution was presented to replace the `ChrootDirectory` feature, Docker containerization along with feature `macvlan bridge`. This allows the traffic flow through the main interface of our Honeypot solution eth0 and Docker subsequently routes the traffic to the designed container using the MAC address assigned to it. The container acts as if it was physically attached to the network. This is done simply by the following command shown in Listing 4.1:

Listing 4.1: Create Docker macvlan bridge.

```
root@template:$ docker network create -d macvlan \
  --subnet= subnet_IP/netmask \
  --gateway= gateway_IP \
  -o parent= name_of_existing_interface \
  name_of_macvlan
```

Furthermore, modifying the `subnet`, `gateway`, and `parent` values to fit the environment in which the macvlan is being created. This creates a macvlan network called `name_of_macvlan`.

Afterwards, the Honeypot container can be connected to the created macvlan network by following command, shown in Listing 4.2:

Listing 4.2: Connect Docker macvlan to the Docker container.

```
root@template:$ docker network connect name_of_macvlan
   docker_container
```

Furthermore, Docker containerization facilitates namespace for the root filesystem on multiple levels, network namespace allowing the container´s own network stack dedicated to it, UTS providing dedicated hostname, IPC proving dedicated shared memory. These traits, of the Docker containerization, provide a significantly improved isolation in comparison to `chroot`.

After all the issues were resolved, a final structure of the new High-interaction Honeypot was put together and subsequently tested on a Virtual Machine, this whole structure with the respected parts is illustrated in the following Fig. 4.2.

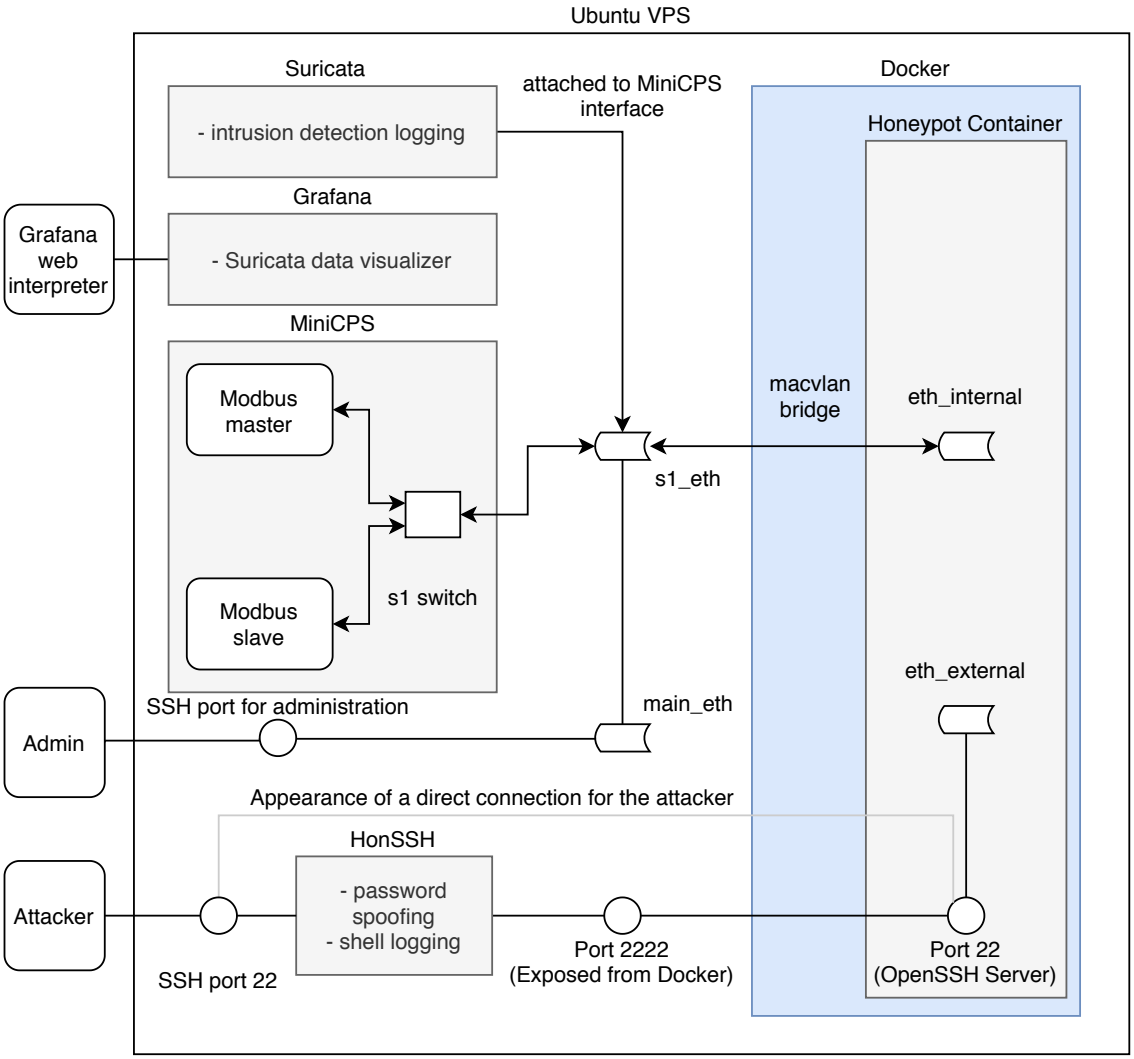

Fig. 4.2: High-interaction Honeypot diagram

4.1.2 Honeypot deployment evaluation

As previously mentioned, the new High-interaction Honeypot solution was deployed onto a clean Virtual Private Server. As the hosting service, `LeaseWeb.com` was chosen for this purpose, based on recommendations and all around well received and professional customer service [41]. For this Honeypot build a server type `gp-small` package was selected provided with 2 Cores , 4 GB of RAM memory and 60 GB of SSD memory. The processing power of the Virtual Private Server far exceeds the needs of the whole Honeypot solution, however there were issues with memory allocation in the previously selected package, thus an upgrade was purchased to get rid of this issue. To premise the outcome of this High-interaction Honeypot solution, an assumption was made, that the connections established with the deployed Honeypot either successful or unsuccessful, would most likely consist of external

file downloads such as malware or worms and data transfer from the Honeypot´s operation system to the attacker´s machine. This premise however might deviate, if the attacker´s motivation exceeds simple data extraction. This thesis presents the High-interaction data evaluation in three steps, moreover three time periods (1 day, 3 days and one week), each building upon one another. The aim of the first time period is to increase the number of successful connections to the Honeypot, which is increased by providing the HonSSH´s password spoofing `users.cfg` configuration file with passwords which occurrence was the highest, during this evaluated time period. Thankfully, there is no need to list through the HonSSH´s bash logs and pick each inserted password individually, moreover these logs do not even provide the password, but rather by utilizing the password spoofing log, which provides list of passwords used with the highest frequency, and the password insertion logs, that lists each individual session password insertion attempt.

During this first time evaluation period, there was total of 1786 connection attempts. The number consists of all connection attempts, where a significant number out of these connections were repeated attempts from same IP addresses. Out of these 1786 connection attempts, 10 connections were captured carrying additional activity upon the successful login, most of these connections originated from the Netherlands. Curiously some of the HonSSH logs that were included in the bash sessions were empty, the reason of this anomaly is unknown. Furthermore, the origin of some connections was not listed by the HonSSH. After running the IP addresses through `IPstack.com`, the location was shown.

	HonSSH	**IPstack.com**		
IP address	Country	Country	Region	City
51.75.52.118	France	Poland	Lower Silesia	Wroclaw
5.79.106.222	China	Netherlands	North Holland	Diemen
183.82.105.103	India	India	Telangana	Hyderabad
195.3.147.47	Latvia	Latvia	Riga	Riga
103.78.243.136	Malysia	Asia	Tsuen Wan	Tsuen Wan
188.32.222.169	Russian Federation	Russia	Moscow	Moscow
92.63.194.105	Russian Federation	Netherlands	Friesland	Drachten
185.153.196.230	Russian Federation	Moldova	Bender Municipality	Bender
193.105.134.45	Sweden	Sweden	Stockholm	Stockholm
75.150.19.221	USA	US	New Mexico	Las Cruces

Tab. 4.1: GeoIP comparison.

The reason for this HonSSH´s log behavior is unknown as well. Thus, the decision was made to run the unknown IP addresses through an external source to see the origin of the connections, this is illustrated in the Tab. 4.1.

Nevertheless, the origin of these sessions is not of the utmost importance as far as this thesis is considered, moreover the attacks are. Out of the aforementioned connections, there were three that stood out from the intentions of the attacker perspective. First connection listed running processes, than proceeded to download an external file through the Secure copy protocol (SCP), to the `/bin` directory. Afterwards, the attacker removed the `/bin/dhcpd` directory and copied the same directory through the SCP. Furthermore the attacker proceeded to change the access permissions of the `/.ssh` directory, followed by listing the rsa key. This connection was ended upon other copied external files to the `/dev` and `/tmp` directories. This connection is shown in the following Listing 4.3, which is reduced to a reasonable size, yet still able to visualize the attack´s course.

Listing 4.3: First stage, attack 1 bash readout.

```
[POT] honey - 127.0.0.1:2222
[SSH] Incoming Connection from 51.75.52.118:35673 - France
[SSH] Login Successful: root:root
[EXEC2] Opened Channel
[EXEC2]: LC_ALL=C top -bn1
[EXEC2]: scp -t /bin/hy1hufqkf9txb4so23rdkbgn0y
[EXEC2]: LC_ALL=C chattr -i -a /bin/dhpcd
[EXEC2]: LC_ALL=C rm -f /bin/dhpcd
[EXEC2]: scp -t /bin/dhpcd
[EXEC2]: LC_ALL=C /bin/dhpcd
[EXEC2]: LC_ALL=C chmod 700 ~/.ssh
[EXEC2]: LC_ALL=C grep "ssh-rsa RSA_KEY" ~/.ssh/
   authorized_keys
[EXEC2]: scp -t /dev/shm/hy1hufqkf9txb4so23rdkbgn0y
[EXEC2]: scp -t /tmp/hy1hufqkf9txb4so23rdkbgn0y
[EXEC2]: LC_ALL=C rm -f /bin/dhpcd
[EXEC2]: scp -t /bin/dhpcd
[EXEC2] Closed Channel
```

The second interesting attack was started with basic info readout of the machine, such as present interfaces, CPU info and system info print out. Furthermore the attacker tried to printout running processes with pipe `'[Mm]iner'`, followed by a readout of files shown in the following Listing 4.4. Assuming this attacker was searching for some sort of a mining software running.

Listing 4.4: First stage, attack 2 bash readout.

```
[POT] honey - 127.0.0.1:2222
[SSH] Incoming Connection from 188.32.222.169:42220 -
   Russian Federation
[SSH] Login Successful: root:root
[EXEC0] Opened Channel
[EXEC1]: ifconfig
[EXEC2]: uname -a
[EXEC3]: cat /proc/cpuinfo
[EXEC5]: ps -ef | grep '[Mm]iner'
[EXEC6]: ls -la /dev/ttyGSM* /var/spool/sms/* /var/log/smsd
   .log /etc/smsd.conf* /usr/bin/qmuxd /var/
   qmux_connect_socket /etc/config/simman /dev/modem* /var/
   config/sms/*
[EXEC7] Closed Channel
```

This testing period´s primary purpose was to increase the number of spoofed connections to the Honeypot. The configuration file with the passwords to be spoofed was supplemented with passwords retrieved from the `spoof.log` of the HonSSH.

Moving onto the second testing time period, there was in total, 7671 connection attempts out of which 16 with activity that were captured by the HonSSH. This testing period will not show the difference between GeoIP comparison, however the result was the same as in the first time period. Assuming this issue will manifest itself even in the last testing period. After the analysis of the occurred attacks, there were few that were almost a mirror copies of one another, with only one difference, being the originating IP address. We can assume that these attackers use a floating IP and each attack is carried out bearing this changing IP address. These attacks, in addition to originating from the same attackers, resembled the course of the Listing 4.3 with few additions shown in Listing 4.5.

Listing 4.5: Second stage, attack 1 bash readout.

```
[POT] honey3days - 127.0.0.1:2222
[SSH] Incoming Connection from 51.77.135.89:38173 - France
[SSH] Login Successful: root:root
[EXEC2] Opened Channel
[EXEC2]: LC_ALL=C /bin/dhpcd -o de.minexmr.com:4444 -B >/
   dev/null 2>/dev/null
[EXEC2]: LC_ALL=C chattr -i /var/spool/cron/crontabs/root
[EXEC2]: LC_ALL=C pkill -HUP sshd
[EXEC2] Closed Channel
```

After initial inspection, the first line of bash input shows data from a port 4444 being pushed to designated directories, the `de.minexmr.com:4444` website seems to facilitate mining software and the port 4444 is a recommended port for the mining activities. Once again it is assumed that the attacker lists through machines that could potentially mine on background without the admin to notice this activity, this however is a speculation. Furthermore, the `chattr` command, which is used to change Linux filesystem attributes. Followed by `pkill`, which is utilized for sending of signals to processes, to kill the SSH deamon process. The other connection of interest seemed to try to exploit weak SSH credentials to build DDoS botnets, based on a found article that dealt with same issues [42], this is shown in the following Listing 4.6.

Listing 4.6: Second stage, attack 2 bash readout.

```
[POT] honey3days - 127.0.0.1:2222                                    1
[SSH] Incoming Connection from 98.159.99.222:47002 - United          2
    States
[SSH] Login Successful: root:123456                                  3
[SSH] Login was spoofed                                              4
[EXEC0] Opened Channel                                               5
[EXEC0] Command Executed: #!/bin/sh\nPATH=$PATH:/usr/local/          6
   sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin\nwget
   http://98.159.99.222/80\ncurl -O http
   ://98.159.99.222/80\nchmod +x 80\n./80\n
[EXEC1] Command Executed: ls -la /var/run/gcc.pid                    7
[EXEC1] Closed Channel                                               8
```

Last but not least, the third stage of exposure of the Honeypot solution was carried out. During this testing period there was a total of 6327 connection attempts, out of which there were 25 logged by the HonSSH sessions log. After further analysis of these sessions, following findings were revealed. The attack sessions of the third exposure period were the sum of the two previous time periods, with differences in only the IP addresses, as mentioned previously, in the description of the second phase.

From these findings, an assumption was made that, the time periods were most likely insufficient as far as the exposure period is concerned, yet the system servers it´s purpose of capturing the attackers and their activities, in an environment that is provided by the High-interaction Honeypot solution. During this period, there was the Suricata IDS deployed to log the JSON logs of the intrusions, for reason unknown the logs did not include the data required to graph locations of the intrusions. This was most likely caused by malfunction of the GeoIP. An unfortunate circumstance,

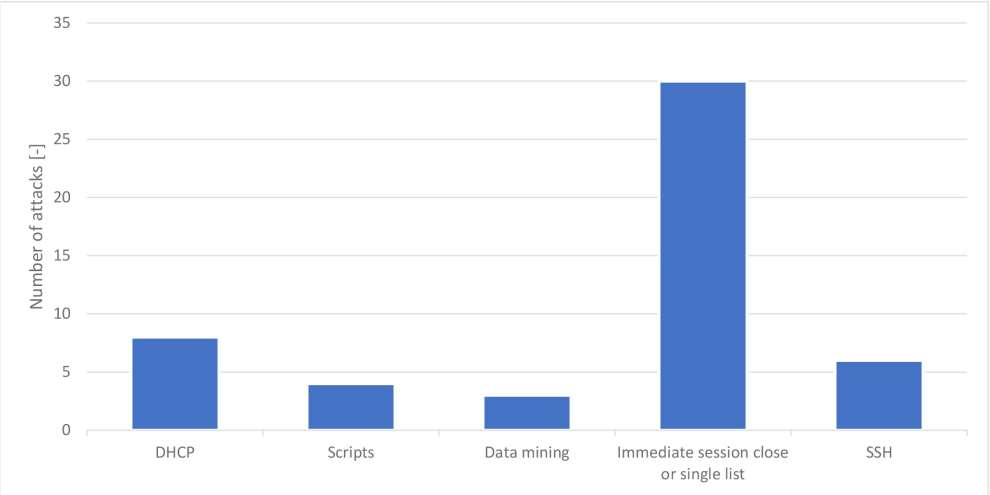

Fig. 4.3: Service attacks distribution.

for the amount of data could provide more insight to the individual attacks. There was almost 50 GB of data captured, this seemed too extensive even for the number of connections, however the amount of data could be caused by the configuration of the Suricata yaml file, for the logging purposes, most of the logging capabilities were enabled. Most attacks of the third testing period were the exact copies of the attacks from previous two periods, mostly bots, script attacks, mining data pushes and ssh exploitation. The sum of the attacks is illustrated in Fig. 4.3.

To visualize the data a different approach was chosen, by utilizing the IP addresses from the HonSSH logs. These addresses were then pushed through the aforementioned `IPstack.com`, from which the geo-locations were extracted. This data was than transformed into a `.xlsx` file, which was imported to the `Google My Maps`. This is illustrated in the Fig. 4.5 and by state in Fig. 4.4.

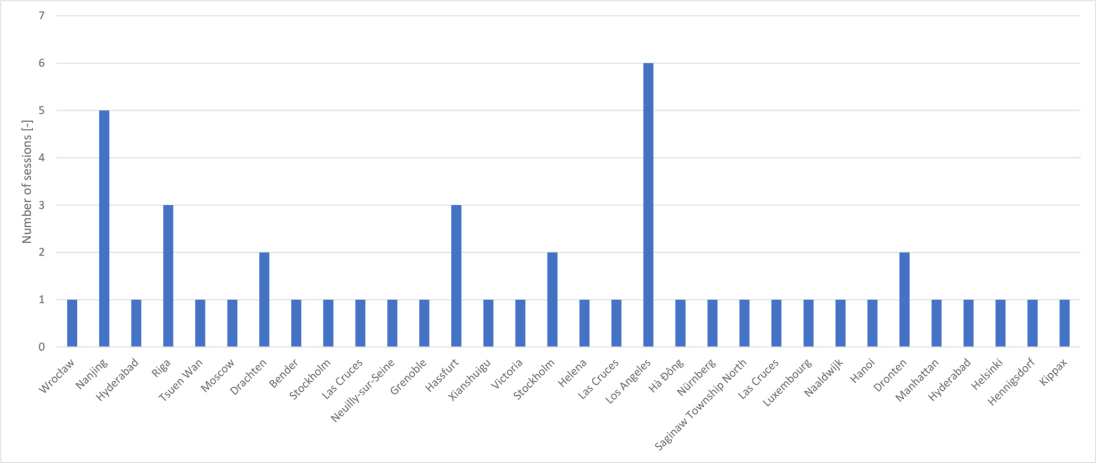

Fig. 4.4: City location of the attacks.

There were other plot solutions tested, however they were either limited to a certain number of locations or not free. This map was cropped to the locations of the attacks origination, for it to be somewhat visible. Still an unresolved issue remains. The inaccuracy of the HonSSH logs in comparison to the subsequent push of the IP addresses through the `IPstack.com`.

To summarize the testing period of the new High-interaction Honeypot solution, the assumption which was made at the beginning of the testing was more or less confirmed, this however does not single out the possibility of the Honeypot solution to be accessed by attacker with intentions of exploiting the ICS capabilities, this would require more extensive testing period however. Nevertheless, the Honeypot solution has proven to be a rather useful tool if used as a part of the local network. Especially, considering the logging capabilities of the attacks that are provided by the HonSSH solution.

Fig. 4.5: Geographic location of the attacks.

Furthermore, this thesis suggests for the deployment to be improve or rather change the password spoofing capabilities, as the initial password login creates an impression of the system being intentionally vulnerable. Another improvement which would most likely convince the attackers as if they have stumbled upon a real machine is to run the Docker container with following attributes, as shown in the following Listing 4.7. This was previously attempted directly in the `/etc/hostname`, this however did not effect the docker hostname. Moreover the issue seems to be that, in order for the container to be named a certain way, it must be defined upon the initiation of the container.

Listing 4.7: Docker hostname set.

```
docker run --name=docker_hostname
```

4.1.3 Honeypot deployment onto a clean Virtual Private Server

This section will be dedicated to the deployment steps of the Honeypot partitions and dependencies onto a clean Virtual Private Server. To access this Virtual Private Server, there are two options presented to us, web console or SSH connection from the administrator machine. For the SSH connection to be successful, a public SSH key needs to be associated to the Virtual Private Server. Firstly, a RSA (Rivest–Shamir–Adleman) key is generated on the administrator´s machine:

Listing 4.8: Generation of a new RSA key.

```
ubuntu@ubuntu-VirtualBox:~/.ssh$ ssh-keygen -t rsa -b 2048
```

A successful generation of the RSA key is displayed by the following readout, Listing 4.9:

Listing 4.9: Successful RSA key generation.

```
The key's randomart image is:
+---[RSA 2048]----+
|OB .  =B..       |
|&.o . o=*.       |
|B+o  . oB .      |
|+O +   + + o     |
|O + .   S o .    |
|+  . .           |
|.   ..           |
| .  . .          |
|  .oo. .E        |
+----[SHA256]-----+
```

Depending on the RSA key strength we can choose a higher bit length, however length of 2048 bits is recommended. Furthermore we need to copy the RSA key to the hosting service. The key is located in the `id_rsa.pub` file. To readout the RSA key, the following Listing 4.10 command can be used:

Listing 4.10: RSA key readout.

```
ubuntu@ubuntu-VirtualBox:~/.ssh$ cat id_rsa.pub
```

Furthermore, SSH associated ports (22, 2222) should be used for the connection with the attacker. For this reason `sshd_config` needs to be adjusted accordingly. This port will be used in conjunction with the public key generated beforehand, moreover this will be an administrative connection.

After the SSH setup is finished, installation of the Honeypot partitions can be initiated. Following steps will describe the installation process of Mininet, MiniCPS,

HonSSH, Docker container for a separate Ubuntu image, Suricata and Grafana service. This section will not describe the installation on Docker, for the process was described in previous part of this thesis, Deployment of Conpot.

In order of the MiniCPS framework to work, MiniNet is required to be installed beforehand. To pull the latest repository of the MiniNet, following Listing 4.11, `git clone` command is used:

Listing 4.11: MiniNet repository pull.

```
root@template:/home/Honey# git clone https://github.com/
    mininet/mininet
```

After a successful image pull, any of the last tagged/released versions of Mininet can be chosen, by listing available versions and defining the source branch, as shown in the following Listing 4.12:

Listing 4.12: MiniNet dependencies installation.

```
root@template:/home/Honey/mininet# git tag
1.0.0
2.0.0
2.1.0
[...]
2.3.0d5
2.3.0d6
cs244-spring-2012-final
root@template:/home/Honey/mininet# git checkout -b 2.3.0d6
Switched to a new branch 'cs244-spring-2012-final'
```

Once the source tree is the defined, the Listing 4.13 command to install Mininet dependencies and packages can be initiated by:

Listing 4.13: MiniNet dependencies and packages installation.

```
root@template:/home/Honey# mininet/util/install.sh -w
[...]
root@template:/home/Honey# sudo apt-get install mininet -a
```

Installation of MiniNet can be triggered with different tags, which effects what parts will be installed. These tags are described in the followigf text.

- ***-a*** - tag installs all partitions included in the Mininet VM, included dependencies such as Open vSwitch along with additions OpenFlow, Wireshark dissector and POX. All of the mentioned tools are installed in the home directory.

- ***-nfv*** -tag installs Mininet, Open vSwitch and OpenFlow reference switch.

- ***-s dir*** - tag installs the additional tools to a selected directory by the user.

After the installation is finished a test to ensure the correct functionality of the MiniNet can be done by following command as shown in Lst 4.14:

Listing 4.14: MiniNet test.

```
root@template:# mn --test pingall                                    1
```

This summarizes the installation of MiniNet framework. Followed by pulling the MiniCPS repository forked from the original `scy-phy/minicps` repository. As shown in the following Listing 4.15:

Listing 4.15: MiniCPS repository pull.

```
root@template:/home/Honey# git clone https://github.com/             1
    Dobrrk/minicps.git
root@template:/home/Honey# pip install minicps                       2
```

This concludes the MiniCPS framework installation, for MiniCPS calls functions from the MiniNet framework and builds on it by implementing it´s own functions as mentioned in Section MiniCPS.

Moving onto the final part of the core structure of the Honeypot solution, `HonSSH` with `twisted cryptography` and `bcrypt` which is a password hashing function. As shown in Listing 4.16.

Listing 4.16: HonSSH pull.

```
root@template:/home/Honey# git clone https://github.com/             1
    tnich/honssh.git
root@template:/home/Honey/honssh# pip install twisted                2
    cryptography --upgrade
root@template:/home/Honey/honssh# pip install bcrypt                 3
```

HonSSH, does not require installation, however it needs to be configured accordingly to the respected interfaces of the hosting Virtual Private Server. This is done by manipulating text fields in the `honssh_cfg` file.

The following steps will define the Docker containerization into which the attacker will be bound upon a new connection. Firstly, image pull of the OS filesystem needs to be carried out as illustrated in the Lst 4.17. The OS filesystem of this thesis choice was Ubuntu 18.04 image located on the `Docker Hub` [43].

Listing 4.17: Docker Ubuntu image pull.

```
root@template:$ docker pull ubuntu                                   1
```

Upon the pull of the Docker image, the container is initialized by command in Lst 4.18, which starts the container, with exposed ssh port.

Listing 4.18: Docker container start.

```
root@template:$ docker run -tid --name honeypot -p
   127.0.0.1:22:2222/tcp ubuntu:18.04 cat
```

The `/tcp` needs to be defined, for if not set only `tcp6`, meaning that the `tcp` SSH service is available only over IP v6.

To access the bash of the container, an SSH connection to localhost can be established or rather, the following command can be initiated as shown in the following Listing 4.19:

Listing 4.19: Connect to the Docker container bash.

```
root@template:$ docker exec -ti honeypot bash
```

Listing 4.20: Docker macvlan bridge set.

```
root@template:$ docker network create -d macvlan --subnet
   =10.0.2.0/24 --gateway=10.0.2.1 -o parent=s1 s1_net
```

Listing 4.21: Docker container connection to the mcvlan.

```
root@template:$ docker network connect s1_net honeypot
```

To increase the Docker container appearance of a frequently used or rather a machine in use, the image should include some of the most frequently used tools for diagnosing the machine or the network. Some of these tools are shown in the Lst 4.22:

Listing 4.22: Tools installation.

```
root@dockercontainer:$ apt-get update \
apt-get install net-tools traceroute iproute2
```

After all of the tools are installed the Docker container is ready for use. In case of a compromise of the Docker container, instead of repeating the process over and over, Docker container´s state can be saved with the `commit` command illustrated in the following Listing 4.23. From this point out, if there are any changes to the container, if run again all changes until the committed version will be kept:

Listing 4.23: Docker commit container version.

```
root@template:$ docker commit honeypot honeypot:
   current_version
```

This concludes the setup of the main structure of the High-interaction Honeypot solution. As a follow up, which is not necessary, but may ease the illustration for the results of the intrusions, a logging Intrusion Detection System, such as Suricata. To install the Suricata IDS, a `.tar` file can be downloaded from the Suricata website through `wget`. After extracting the Suricata, the configuration file `suricata.yaml` needs to be configured according to the system on which it runs. Some of the main settings are the log path, rules applied to the IDS and the output format of the logs depending on the visualization software of choice.

Afterwards the Suricata IDS can be started with the following command show in Lst 4.24:

Listing 4.24: Suricata initialization.

```
root@template:$ suricata_PATH/suricata -c config_PATH/
   suricata.yaml -i eth0 -l /var/log/suricata
```

The last step which is also not necessary is the visualization solution. In this thesis Grafana was chosen based upon a suggestion of it´s ease of use and visualization capabilities. Grafana can be downloaded and extracted from the Grafana web. To access Grafana server, it needs to be run on a designated machine, moreover it does not need to be the machine housing the Honeypot, however it helps in a way of easier data migration. The Grafana server can be accessed on port 3000 through a web browser. This concludes the process of setting up the Honeypot solution onto a clean Virtual Private Server.

Conclusion

The aim of this thesis was to analyze available Honeypot resources for the industrial systems and systems of automation. These systems were to be theoretically analyzed based on their availability, protocol support or level of interaction. To summarize the theoretical chapter, the basics of Industrial Systems were described, and most commonly used abbreviations were described. Followed by IT and OT comparison, protocols used in the OT environment, possible attacks connected with the ICS systems and subsequent description of the Honeypots systems and brief overview of few selected Honeypot systems. Followed by practical deployment and simulation of the chosen Honeypot system. The evaluation consisted of several service session log outputs, their contents, capabilities and shortcomings. During these service tests the data was captured by a third party software to see the impact of the Honeypot systems on the virtualized machine itself.

Moving onto the prospects of further continuation and expansion of this thesis. As mentioned in the practical sections of this thesis, more comprehensive tests could be carried out, such as malware attacks, brute force attacks, Man in the Middle attacks or injection type attacks. The main goals are to chose the most suitable system that will be implemented into a VUT infrastructure and put into operation opting for either low or high interaction level. Subsequent evaluation of the data of the Honeypot system should take place after a longer period of time, based on the log findings.

There should also be an analysis conducted based on the findings of the deployed Honeypot system, concerning what ports and services are targeted primarily. Based on this we could point out the short comings of the current state of the Honeypot system extensions and to build upon and scale up the current structure of the system. The main focus will be dedicated to the development of a high interaction Honeypot system for ICS systems abuse protection.

There is a motivation to explore high interaction Honeypot systems, for they were not included in the practical part of this thesis. Furthermore, to deploy a Honeypot system for an extended period of time to capture foreign attack's data and subsequently to evaluate them. Initially, the Conpot Honeypot seems to be the best option for this, yet the Cowrie Honeypot might provide some interesting data as well. Furthermore, we would like to develop a new Industrial Control System device which would be emulated by the Conpot Honeypot. This remains for a subsequent discussion concerning what device would fit into the VUT infrastructure with the protocols that should be supported and capabilities which should be utilized by the emulated device. This will require analysis of the VUT infrastructure as well as the data captured from the various attacks captured by the deployed Honeypot systems.

Upon the changes made in the assignment of the thesis, the deployment of the Industrial Control System High-interaction Honeypot solution was to be made externally, the decision was to deploy it to a hosting service `Leaseweb.com`, for the positive reviews and exceptional customer´s service. The deployed ICS Honeypot solution provides two implemented protocols, Modbus TCP and EtherNet/IP running on the MiniCPS platform. The whole system utilizes multiple parts, MiniCPS that simulates the physical processes, MiniNet on which the MiniCPS builds upon for the network topology simulation, HonSSH as a bash logging solution and finally the Docker containerization for the SSH traffic to be forwarded to, which subsequently traps the intruders or rather isolates then from exploiting the whole machine. From the Docker container, the intruders are presented with an Ubuntu kernel, processes running from the MiniCPS. If the container is compromised, it is easily recoverable from a previously committed image of the container. The Docker container exposes SSH port 2222 to the main hosting machine, to which the HonSSH is connected, furthermore the HonSSH exposes the port 22 to the Internet, and all connections are pushed through it to the container.

The High-interaction Honeypot solution was tested in three exposure periods. The first period´s main purpose was to increase the number of successful connections to the Honeypot, thanks to the HonSSH ability of password spoofing based either on a list of passwords in the configuration file or the probability of a successful connection. Either option seems liable. During this exposure period there were few sessions of interest. These sessions included downloads external files and forceful removal of several OS directories. The second period included attacks trying to push mining data from external website, downloads of scripts and permission changes. The third period included attacks that were the sum of the two previous periods. There were no attacks that would include the intention of ICS device exploit. A conclusion was made that the Honeypot solution serves it´s purpose even for no recorded ICS attack sessions, this could however change if the exposure periods were longer. Nevertheless, the Honeypot described of this thesis seems to be a liable solution and a useful addition to a network, especially for the log capabilities.

Bibliography

[1] Cybercrimemag. Cybercrime damages 6 trillion by 2021, Dec 2018. URL: https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/.

[2] Yacov Y Haimes. *Modeling and managing interdependent complex systems of systems*. John Wiley & Sons, 2018.

[3] Tyson Macaulay and Bryan L Singer. *Cybersecurity for industrial control systems: SCADA, DCS, PLC, HMI, and SIS*. Auerbach Publications, 2016.

[4] Wolfgang Schwab and Mathieu Poujol. The state of industrial cybersecuity 2018. *Trend Study Kaspersky Reports*, page 33, 2018.

[5] Ot, ics, scada – what's the difference? URL: https://www.kuppingercole.com/blog/williamson/ot-ics-scada-whats-the-difference.

[6] Bywebranded. Information technologies vs operational technologies, Mar 2019. URL: www.randed.com.

[7] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to industrial control systems (ics) security. *NIST special publication*, 800(82):16–16, 2011.

[8] Peter Huitsing, Rodrigo Chandia, Mauricio Papa, and Sujeet Shenoi. Attack taxonomies for the modbus protocols. *International Journal of Critical Infrastructure Protection*, 1:37–44, 2008.

[9] Modbus tcp/ip for ethernet i/o, Aug 2019. URL: https://acromag.com/modbus-tcp-ip-for-ethernet-i-o/.

[10] Paul Brooks. Ethernet/ip-industrial protocol. In *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 01TH8597)*, volume 2, pages 505–514. IEEE, 2001.

[11] Softprom.com. Cyber attacks on critical infrastructure. URL: https://softprom.com/.

[12] Joseph Slowik. Evolution of ics attacks and the prospects for future disruptive events.

[13] Michael J Assante and Robert M Lee. The industrial control system cyber kill chain. *SANS Institute InfoSec Reading Room*, 1, 2015.

[14] Defending ics and scada systems from cyber attacks, Oct 2017. URL: `www.iiot-world.com`.

[15] Fred Cohen et al. The deception toolkit. *Risks Digest*, 19:1998, 1998.

[16] Cliff Stoll. *The cuckoo's egg: tracking a spy through the maze of computer espionage*. Simon and Schuster, 2005.

[17] Honeynet Project. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Addison-Wesley Professional, 2001.

[18] Feng Zhang, Shijie Zhou, Zhiguang Qin, and Jinde Liu. Honeypot: a supplemented active defense system for network security. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 231–235. IEEE, 2003.

[19] Know your Enemy. Malicious web servers. *Christian Seifert, Ramon Steenson, Ian Welch, Peter Komisarczuk, Thorsten Holz, Bing Yuan, Michael A. Davis*, 2007.

[20] Manuel Egele, Engin Kirda, Christopher Kruegel, and Peter Wurzinger. Defending browsers against drive-by downloads. *DIMVA 2009, July 9-10, 2009, Milan, Italy*, 2009.

[21] Jason Nieh and Ozgur Can Leonard. Examining vmware. *Dr. Dobb's Journal*, 25(8):70, 2000.

[22] Oracle VM Virtualbox. Oracle vm virtualbox. *Change*, 107:1–287, 2011.

[23] Matthew Ian Hepburn. Server message block (smb) security signatures seamless session switch, August 14 2012. US Patent 8,245,287.

[24] Eric Peter and Todd Schiller. A practical guide to honeypots. *Washington Univerity*, 2011.

[25] Iyatiti Mokube and Michele Adams. Honeypots: concepts, approaches, and challenges. In *Proceedings of the 45th annual southeast regional conference*, pages 321–326. ACM, 2007.

[26] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[27] Adam Schoeman. Amber: A zero-interaction honeypot and network enforcer with modular intelligence. In *2013 Information Security for South Africa*, pages 1–7. IEEE, 2013.

[28] David Moore, Colleen Shannon, Geoffrey M Voelker, and Stefan Savage. Network telescopes. Technical report, Technical Report CS2004-0795, CSE Department, UCSD, 2004.

[29] Maya Bercovitch, Meir Renford, Lior Hasson, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Honeygen: An automated honeytokens generator. In *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, pages 131–136. IEEE, 2011.

[30] Daniele Antonioli and Nils Ole Tippenhauer. Minicps: A toolkit for security research on cps networks. In *Proceedings of the First ACM workshop on cyber-physical systems-security and/or privacy*, pages 91–100, 2015.

[31] Daniele Antonioli, Anand Agrawal, and Nils Ole Tippenhauer. Towards high-interaction virtual ics honeypots-in-a-box. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pages 13–22, 2016.

[32] Mininet Team. Mininet, 2018. URL: `http://mininet.org/`.

[33] Mininet. mininet/mininet. URL: `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`.

[34] Thomas Nicholson. Honssh, high-interaction honeypot solution, Apr 2018. URL: `https://github.com/tnich/honssh`.

[35] Thomas Nicholson. Honssh - advanced networking.

[36] Suricata Open Source IDS. Ips/nsm engine. *DOI: http://suricata-ids. org*.

[37] Peter Soóky. Extented functionality of honeypots. *bachelor's thesis*, 2015.

[38] Galen Collins. Pymodbus documentation, 2013.

[39] Perry Kundert. Cpppo library. URL: `https://github.com/pjkundert/cpppo`.

[40] Random - generate pseudo-random numbers. URL: `https://docs.python.org/3/library/random.html`.

[41] Global hosted infrastructure (iaas) and cloud solutions. URL: `https://www.leaseweb.com/`.

[42] Grehack international security conference. URL: `https://grehack.fr/`.

[43] Docker hub ubuntu image repository. URL: `https://hub.docker.com/_/ubuntu/`.

List of symbols, physical constants and abbreviations

ICS	Industrial Control System
IT	Information Technology
OT	Operational Technology
SCADA	Supervisory Control and Data Acquisition systems
PLC	Programmable Logic Controller
HMI	Human Machine Interface
PCD	Process Control Dynamics
HTTP	Hypertext Transfer Protocol
SNMP	Simple Network Management Protocol
IMPI	Intelligent Platform Management Interface
ENIP	EtherNet/IP (Industrial Protocol)
FTP	File Transfer Protocol
TFTP	Trivial File Transfer Protocol
URI	Uniform Resource Identifier
SSH	Secure Shell
IP address	Internet Protocol address